COURSE GUIDE

IFT 203
INTRODUCTION TO WEB TECHNOLOGIES

Course Team Dr. Awotunde J. B. (Course Developer/Writer)-UI
Prof Peter Ogedebe (Course Editor)-
Base University, Abuja

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
University Village
Plot 91, Cadastral Zone
Nnamdi Azikiwe Expressway
Jabi, Abuja

Lagos Office
14/16 Ahmadu Bello Way
Victoria Island, Lagos

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Printed 2025

ISBN: 978-978-786-487-6

CONTENTS

Introduction

IFT 203 – Introduction to Web Technologies is a two-credit unit course. It deals with the introduction of the Internet, the World Wide Web (WWW), and Web development. WWW is a platform for interactive applications, content publishing, and social services. The role of HTTP and HTTPS in the context of web applications. Roles and operations of web browsers and the web server. Interacting with web applications through forms, and using style sheets to separate document structure and document formatting.

Course Competencies

This course equips students with foundational skills in web development and design. Key competencies include understanding the basic principles of the internet and web technologies, proficiency in HTML and CSS for creating and styling web pages, and an introduction to JavaScript for adding interactivity. Additionally, the course covers web standards and best practices, responsive design techniques, and an awareness of accessibility considerations. Students will also learn about web hosting, domain names, and basic SEO principles, enabling them to create functional, user-friendly websites from scratch.

Course Objectives

Certain objectives have been set out to ensure that the course achieves its aims. Apart from the general objectives of this course, each unit of this course has set objectives. At the end of this course, you should be able to:

- discuss the evolution of the Internet and explain the meaning of Intranet and extranet
- list the devices used to access the Internet and explain the various means of accessing the Internet • differentiate between static and dynamic pages
- describe the term “computer network,” discuss the client-server model, and describe the Web application architecture
- explain the term “HTML,” write simple HTML codes using popular tags, and use Web browsers to display HTML codes
- explain the term “XHTML,” write simple HTML5 codes using popular tags, and use Web browsers to display XTML codes
- outline how to create, modify, process, view, and validate XML document
- comprehend how the internet works, including protocols such as HTTP and HTTPS.
- understand the client-server model and the role of web browsers and servers.

- gain proficiency in HTML for structuring web content.
- master CSS for styling and layout of web pages.
- implement responsive design techniques using CSS media queries.
- Learn about W3C standards and their importance in web development.
- understand the principles of clean, semantic HTML and CSS coding.
- Discuss the importance of CSS, use CSS format web pages, and add CSS to HTML files
- explain the meaning of JavaScript
- Write and run simple JavaScript programs

Working Through this Course

This course provides an overview of the fundamental concepts and technologies that drive the World Wide Web. Students will gain practical experience in web development, learning how to create and manage web content, understand web protocols, and work with various web technologies. Topics include HTML, CSS, JavaScript, web hosting, and basic principles of web design and user experience (UX). To have a thorough understanding of the course units, you will need to read and understand the contents, practice what you have learned by studying and developing simple websites and Web applications for your organization, and be committed to learning and using skills acquired from the course to enhance your career.

Study Units

This course has four modules broken down into fifteen (15) study units. They are listed as follows:

Module 1 Introduction to the World Wide Web

Unit 1 History of the Internet and the Web
Unit 2 Understanding Web Browsers
Unit 3 Internet Services, Communication and Protocol
Unit 4 Network model and web application development

Module 2 HTML Fundamentals

Unit 1 Introduction to HTML
Unit 2 HTML tags and attributed
Unit 3 HTML syntax and basic markup: headings, paragraphs, lists, links
Unit 4 Advanced HTML Markup

Module 3 Cascading Style Sheet (CSS)Basics

Unit 1 Introduction to Cascading Style Sheet
Unit 2 Styling with Cascading Style Sheet

Module 4 Introduction to JavaScript

Unit 1 Basics of JavaScript
Unit 2 Fundamentals of JavaScript for Dynamic Statements
Unit 3 Document Object Model (DOM) Manipulation

References and Further Readings

Duckett, J. (2011). HTML & CSS: design and build websites (Vol. 15). Indianapolis, IN, USA:: Wiley.

Duckett, J. (2014). Javascript and jquery: Interactive front-end web development. Wiley Publishing.

Robbins, J. N. (2012). Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. " O'Reilly Media, Inc.".

Haverbeke, M. (2018). Eloquent javascript: A modern introduction to programming. No Starch Press.

Frain, B. (2012). Responsive web design with HTML5 and CSS3. Packt Publishing Ltd.

Frain, B. (2015). Responsive web design with HTML5 and CSS3. Packt Publishing Ltd.

Felke-Morris, T. (2015). Web development and design foundations with HTML5 (p. 672). Pearson education limited.

Sorathiya, D. (2014). Learn java script. Dharm Sorathiya.

Nixon, R. (2014). Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. " O'Reilly Media, Inc.".

Idowu, S.A., Maitanmi, S.O. & Adetunji, O. O. (2020). Introductory to Web Technology and Development. Nigeria: Jamiro Press.

Nagpal, D. P. (2006). *Web Design Technology, Theory and Technique on the Cutting Edge.* New Delhi, India: S. Chand and Company Ltd.

Shklar, L. & Rosen, R. (2009). Web Application Architecture, Principles, Protocols and Practices. England: John Wiley & Sons Ltd.

Wang, P., & Katila, S. (2003). An Introduction to Web Design and Programming. Brooks/Cole book/

Nolan, H. (2005). *Creating a Web Page in Dreamweaver*. USA: Peachpit Press, Berkeley.

MacDonald, M. (2013). HTML5: The missing manual. " O'Reilly Media, Inc.".

Presentation Schedule

The presentation schedule included in your course materials gives you the important dates for the completion of tutor-marked assignments and attending tutorials. Remember, you are required to submit all your assignments by the due date. You should guard against lagging in your work.

Assessment

There are two aspects to the assessment of the course. First are the tutor-marked assignments; second is a written examination. In tackling the assignments, you are expected to apply information and knowledge acquired during this course. The assignments must be submitted to your tutor for formal assessment following the deadlines stated in the Assignment File. The work you submit to your tutor for assessment will count for 30 percent of your total course mark. At the end of the course, you will need to sit for a final three-hour examination. This will also count for 70 percent of your total course mark.

How to Get the Most from the Course

In distance learning, the study units replace the university lecturer. This is one of the great advantages of distance learning; you can read and work through specially designed study materials at your own pace, and at a time and place that suit you best. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some reading to do, the study units tell you when to read your textbooks or other material. Just as a lecturer might give you an in-class exercise, your study units provide exercises for you to do at appropriate points.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next is a set of learning objectives. These objectives enable you to know what you should be able to do by the time you have completedthe unit. You should use these objectives to guide your study. When you havefinished the units, you must go back and check whether you have achieved the objectives. If you make a habit of doing this, you will significantly improve your chances of passing the course.

Remember that your tutor's job is to assist you. When you need help, do not hesitate to call and ask your tutor to provide it.

1. Read this Course Guide thoroughly.
2. Organize a study schedule. Refer to the „Course Overview" for more details. Note the time you are expected to spend on each unit and how the assignments relate to the units. Whatever method you chose to use, you should decide on it and write in your dates for working on each unit.
3. Once you have created your study schedule, do everything you can to stick to it. The major reason that students fail is that they lag in their coursework.
4. Turn to Unit 1 and read the introduction and the objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the „Overview" at the beginning of each unit. You will almost always need both the study unit you are working on and one of your sets of books on your desk at the same time.
6. Work through the unit. The content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be instructed to read sections from your set books or other articles. Use the unit to guide your reading.
7. Review the objectives for each study unit to confirm that you have achieved them. If you feel unsure about any of the objectives, review the study material or consult your tutor.
8. When you are confident that you have achieved a unit's objectives, you can then start on the next unit. Proceed unit by unit through the course and try to pace your study so that you keep yourself on schedule.
9. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor-marked assignment form and written on the assignment. Consult your tutor as soon as possible if you have any questions or problems.
10. After completing the last unit, review the course and prepare

yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this Course Guide).

Facilitation

There are 12 hours of tutorials provided in support of this course. You will be notified of the dates, times, and locations of the set tutorials, together with the name and phone number of your tutor, as soon as you are allocated a tutorial group. Do not hesitate to contact your tutor by telephone, e-mail, or discussion board if you need help. You will benefit a lot by doing that. Contact your tutor if:

- You do not understand any part of the study units or the assigned readings
- You have difficulty with the self-tests or exercises
- you have a question or problem with an assignment, with your tutor's comments on an assignment, or with the grading of an assignment.

You should try to attend the tutorials. Thus, it is the only opportunity you have to enjoy face-to-face contact with your tutor and to ask questions that are answered instantly. You can raise any problem encountered during your study. To gain the maximum benefit from course tutorials, prepare a question list before attending them. You will learn a lot from participating in the discussion actively.

Course Information

Course Code	IFT 203
Course Title	Introduction to Web Technology
Credit Unit	2
Course Status	Compulsory

Course Blub:

This course dives into the dynamic world of web development with a comprehensive introduction to web technologies. Designed for beginners, this course provides a foundational understanding of the tools and technologies that power the modern web. From HTML and CSS to JavaScript and beyond, you'll gain hands-on experience building and deploying web pages and applications. Learn the basics of HTML to structure your web content and CSS to style and layout your pages. Discover the power of JavaScript for creating interactive and dynamic web experiences. Understand the principles of responsive web design to ensure your sites work seamlessly across various devices. Gain a preliminary understanding of server-side technologies and how they interact with the client side. This course is perfect for aspiring web developers, designers, or anyone interested in learning how to create websites. No prior programming experience is required, just a passion for learning and a curiosity about how the web works. The course includes a mix of video lectures, interactive coding exercises, quizzes, and hands-on projects. By the end of the course, you'll have built your own functional website and gained the confidence to further explore advanced web development topics.

Semester:	first
Course Duration:	13 weeks
Required Hours for Study:	45

MAIN COURSE

CONTENTS

MODULE 1 INTRODUCTION TO THE WORLD WIDE WEB

MODULE INTRODUCTION

The module "Introduction to the World Wide Web" offers a comprehensive overview of the Internet's cornerstone technology, enabling students to grasp the foundational concepts and historical evolution of the web. This module begins by exploring the origins and development of the World Wide Web, tracing its transformation from a research project at CERN to the global information infrastructure it is today. Students will gain insight into web browsers, including their various types. Additionally, the module delves into internet services, communication, and protocol. As the module progresses, it addresses the practical aspects of web development and usage. Learners will engage in hands-on activities designed to build their skills in designing, developing, and deploying web pages. By the end of the course, students will have a solid foundation in web technologies, preparing them for further study or careers in web development, digital marketing, or related fields.

Unit 1 History of the Internet and the Web

Contents

1.0 Introduction

The World Wide Web (WWW) allows computer users to position and view multimedia-based documents (i.e., documents with text, graphics, animations, audio, and/or videos) on almost any subject. Even though the Internet was developed more than three decades ago, the introduction of the WWW was a relatively recent event. In 1990, Tim Berners-Lee of CERN (the European Laboratory for Particle Physics) developed the World Wide Web and several communication protocols that form the backbone of the WWW. The Internet and the World Wide Web will surely be listed among the most significant and profound creations of humankind. In the past, most computer applications ran on stand-alone computers. (i.e., computers that were not connected) Today's applications can be written to communicate among the world's hundreds of millions of computers. The Internet makes our work easier by mixing computing and communications technologies. It makes information immediately and conveniently accessible worldwide. It makes it possible for individuals and small businesses to get worldwide contact. In the last decade, the Internet and the World Wide Web have altered the way people communicate, conduct business, and manage their daily lives. They are changing the nature of the way business is done.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Understand ARPANET and the Birth of the Internet.
- Explain the Tim Berners-Lee and the Invention of the World Wide Web
- Explain the evolution of the web: from static to dynamic content

3.0 Main Content

3.1 Introduction to ARPANET and the birth of the internet

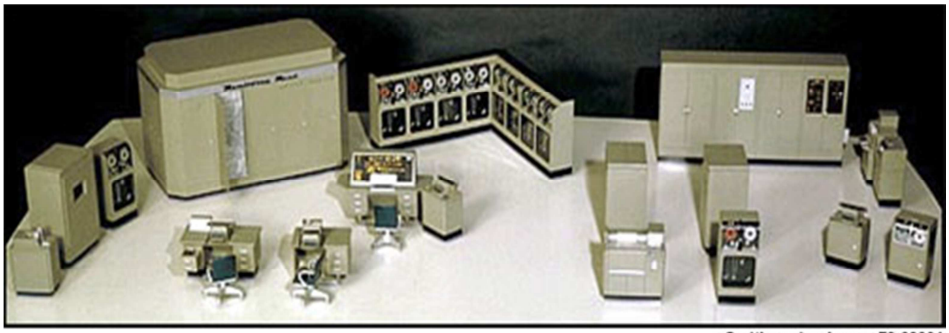
The simplest way of explaining the Internet is to call it "the network of networks." It's the connection of computer networks around the world into one entity, so to speak. It's not one big computer, but rather numerous networked computers connected together.

When you dial into your Internet service provider (AOL, Earthlink, etc) from home, you are essentially connecting your computer to a network. If you are on campus you connect to the Internet through your school's network, which is connected to the larger Internet network through Peachnet, which is the electronic highway for all educational institutions and libraries throughout the state of Georgia. The "backbone" of all these connections is what you might hear referred to as the "information superhighway."

The Internet started in the 1960s as a way for government researchers to share information. Computers in the '60s were large and immobile and in order to make use of information stored in any one computer, one had to either travel to the site of the computer or have magnetic computer tapes sent through the conventional postal system.

Another catalyst in the formation of the Internet was the heating up of the Cold War. The Soviet Union's launch of the Sputnik satellite spurred the U.S. Defense Department to consider ways information could still be disseminated even after a nuclear attack. This eventually led to the formation of the ARPANET (Advanced Research Projects Agency Network), the network that ultimately evolved into what we now know as the Internet. ARPANET was a great success but membership was limited to certain academic and research organizations who had contracts with the Defense Department. In response to this, other networks were created to provide information sharing.

January 1, 1983 is considered the official birthday of the Internet. Prior to this, the various computer networks did not have a standard way to communicate with each other. A new communications protocol was established called Transfer Control Protocol/Internetwork Protocol (TCP/IP). This allowed different kinds of computers on different networks to "talk" to each other. ARPANET and the Defense Data Network officially changed to the TCP/IP standard on January 1, 1983, hence the birth of the Internet. All networks could now be connected by a universal language.



Smithsonian Image 73-03061

The image above is a scale model of the UNIVAC I (the name stood for Universal Automatic Computer) which was delivered to the Census Bureau in 1951. It weighed some 16,000 pounds, used 5,000 vacuum tubes, and could perform about 1,000 calculations per second. It was the first American commercial computer, as well as the first computer designed for business use. (Business computers like the UNIVAC processed data more slowly than the IAS-type machines, but were designed for fast input and output.) The first few sales were to government agencies, the A.C. Nielsen Company, and the Prudential Insurance Company. The first UNIVAC for business applications was installed at the General Electric Appliance Division, to do payroll, in 1954. By 1957 Remington-Rand (which had purchased the Eckert-Mauchly Computer Corporation in 1950) had sold forty-six machines.
ARPANET, an experimental computer network that was the forerunner of the Internet. The Advanced Research Projects Agency (ARPA), an arm of the U.S. Defense Department, funded the development of the Advanced Research Projects Agency Network (ARPANET) in the late 1960s. Its initial purpose was to link computers at Pentagon-funded research institutions over telephone lines.

At the height of the Cold War, military commanders were seeking a computer communications system without a central core, with no headquarters or base of operations that could be attacked and destroyed by enemies thus blacking out the entire network in one fell swoop. ARPANET's purpose was always more academic than military, but, as more academic facilities connected to it, the network did take on the tentacle-like structure military officials had envisioned. The Internet essentially retains that form, although on a much larger scale.

Roots of a network
ARPANET was an end-product of a decade of computer-communications developments spurred by military concerns that the Soviets might use their jet bombers to launch surprise nuclear attacks against the United States. By the 1960s, a system called SAGE (Semi-Automatic Ground Environment) had already been built and was using computers to track incoming enemy aircraft and to coordinate military response. The system included 23 "direction centers," each with a massive mainframe computer that could track 400 planes, distinguishing friendly aircraft from enemy bombers. The system required six years and $61 billion to implement.

The system's name hints at its importance, as author John Naughton points out. The system was only "semi-automatic," so human interaction was pivotal. For Joseph Carl Robnett Licklider, who would became the first director of ARPA's Information Processing Techniques

Office (IPTO), the SAGE network demonstrated above all else the enormous power of interactive computing—or, as he referred to it in a seminal 1960 essay, of "man-computer symbiosis." In his essay, one of the most important in the history of computing, Licklider posited the then-radical belief that a marriage of the human mind with the computer would eventually result in better decision-making.

In 1962, Licklider joined ARPA. According to Naughton, his brief two-year stint at the organization seeded everything that was to follow. His tenure signaled the demilitarization of ARPA; it was Licklider who changed the name of his office from Command and Control Research to IPTO. "Lick," as he insisted on being called, brought to the project an emphasis on interactive computing and the prevalent utopian conviction that humans teamed with computers could create a better world.

Perhaps in part because of Cold War fears, during Licklider's IPTO tenure, it is estimated that 70 percent of all U.S. computer-science research was funded by ARPA. But many of those involved said that the agency was far from being a restrictive militaristic environment and that it gave them free rein to try out radical ideas. As a result, ARPA was the birthplace not only of computer networks and the Internet but also of computer graphics, parallel processing, computer flight simulation, and other key achievements.

Ivan Sutherland succeeded Licklider as IPTO director in 1964, and two years later Robert Taylor became IPTO director. Taylor would become a key figure in ARPANET's development, partly because of his observational abilities. In the Pentagon's IPTO office, Taylor had access to three teletype terminals, each hooked up to one of three remote ARPA-supported time-sharing mainframe computers—at Systems Development Corp. in Santa Monica, at UC Berkeley's Genie Project, and at MIT's Compatible Time-Sharing System project (later known as Multics).

In his room at the Pentagon, Taylor's access to time-shared systems led him to a key social observation. He could watch as computers at all three remote facilities came alive with activity, connecting local users. Time-shared computers allowed people to exchange messages and share files. Through the computers, people could learn about each other. Interactive communities formed around the machines.
Taylor also decided that it made no sense to require three teletype machines just to communicate with three incompatible computer systems. It would be much more efficient if the three were merged into one, with a single computer-language protocol that could allow any

terminal to communicate with any other terminal. These insights led Taylor to propose and secure funding for ARPANET.

A plan for the network was first made available publicly in October 1967, at an Association for Computing Machinery (ACM) symposium in Gatlinburg, Tennessee. There, plans were announced for building a computer network that would link 16 ARPA-sponsored universities and research centers across the United States. In the summer of 1968, the Defense Department put out a call for competitive bids to build the network, and in January 1969 Bolt, Beranek, and Newman (BBN) of Cambridge, Massachusetts, won the $1 million contract.

According to Charles M. Herzfeld, the former director of ARPA, Taylor and his colleagues wanted to see if they could link computers and researchers together. The project's military role was much less important. But at the time it was launched, Herzfeld noted, no one knew whether it could be done, so the program, initially funded on $1 million diverted from ballistic-missile defense, was risky.

Taylor became ARPA's computer evangelist, picking up Licklider's mantle and preaching the gospel of distributed interactive computing. In 1968, Taylor and Licklider co-authored a key essay, "The Computer as a Communication Device," which was published in the popular journal *Science and Technology*. It began with a thunderclap: "In a few years, men will be able to communicate more effectively through a machine than face to face." The article went on to predict everything from global online communities to mood-sensing computer interfaces. It was the first inkling the public ever had about the potential of networked digital computing, and it attracted other researchers to the cause.

A packet of data

ARPANET arose from a desire to share information over great distances without the need for dedicated phone connections between each computer on a network. As it turned out, fulfilling this desire would require "packet switching."

Paul Baran, a researcher at the RAND Corporation think tank, first introduced the idea. Baran was instructed to come up with a plan for a computer communications network that could survive nuclear attack and continue functioning. He came up with a process that he called "hot-potato routing," which later became known as packet switching.

"Packets" are small clusters of digital information broken up from larger messages for expediency's sake. To illustrate in more recent terms: an e-mail might be split into numerous electronic packets of information and transmitted almost at random across the labyrinth of the nation's

telephone lines. They do not all follow the same route and do not even need to travel in proper sequential order. They are precisely reassembled by a modem at the receiver's end, because each packet contains an identifying "header," revealing which part of the larger message it represents, along with instructions for reconstituting the intended message. As a further safeguard, packets contain mathematical verification schemes that insure data does not get lost in transit. The network on which they travel, meanwhile, consists of computerized switches that automatically forward packets on to their destination. Data packets make computer communications more workable within existing telephone infrastructure by allowing all those packets to flow following paths of least resistance, thereby preventing logjams of digital data over direct, dedicated telephone lines.

Baran's idea was ignored by the military. A 1964 paper outlining his innovation was published, but it was classified and began to collect dust. Fortunately, one place it was collecting dust was in the offices of ARPA, where it was eventually rediscovered. Baran's idea became the key concept that made ARPANET possible. Packet-switched communication remains perhaps the most important legacy handed down to the Internet by ARPANET.

Rise and fall

In late 1969, a team of UCLA graduate students under the leadership of professor Leonard Kleinrock sent the first packet-switched message between two computers. A member of Kleinrock's team, Charley Kline, had the distinction of being first to send it, but it was not a rousing start. As Kline at UCLA tried logging into the Stanford Research Institute's computer for the first time, the system crashed just as he was typing the letter "G" in "LOGIN."

The bugs were worked out, and further connections were made flawlessly, but the early network had many limitations. At the time of Kline's first message to Stanford, logging into a remote computer was one of just three tasks possible on ARPANET; the other options were printing to a remote printer and transferring files between computers. Nevertheless, the interest generated by the nascent two-node network was intense. By the end of 1969, academic institutions were scrambling to connect to ARPANET. The University of California–Santa Barbara and the University of Utah linked up that year. By April 1971, there were 15 nodes and 23 host terminals in the network. In addition to the four initial schools, contractor BBN had joined, along with MIT, the RAND Corporation, and NASA, among others. By January 1973, there were 35 connected nodes; by 1976, there were 63 connected hosts.

During its first 10 years, ARPANET was a test bed for networking innovations. New applications and protocols like Telnet, file transfer protocol (FTP), and network control protocol (NCP) were constantly being devised, tested, and deployed on the network. In 1971, BBN's Ray Tomlinson wrote the first e-mail program, and the ARPANET community took to it instantly. "Mailing lists," which eventually became known as "LISTSERVs," followed e-mail almost immediately, creating virtual discussion groups. One of the first e-mail discussion lists was SF-LOVERS, which was dedicated to science fiction fans.

What ARPANET could not do was talk to any of the other computing networks that inevitably sprang up in its wake. Its design required too much control and too much standardization among machines and equipment on the network, according to Naughton. So in the spring of 1973, Vinton Cerf and Bob Kahn began considering ways of connecting ARPANET with two other networks that had emerged, specifically SATNET (satellite networking) and a Hawaii-based packet radio system called ALOHANET. One day, waiting in a hotel lobby, Cerf dreamed up a new computer communications protocol, a gateway between networks, which eventually became known as transmission-control protocol/Internet protocol (TCP/IP). TCP/IP, which was first tested on ARPANET in 1977, was a way that one network could hand off data packets to another, then another, and another. Eventually, when the Internet consisted of a network of networks, Cerf's innovation would prove invaluable. It remains the basis of the modern Internet.

In 1975, ARPANET was transferred to the Defense Communications Agency. By that time, it was no longer experimental, nor was it alone. Numerous new networks had emerged by the late 1970s, including CSNET (Computer Science Research Network), CDnet (Canadian Network), BITNET (Because It's Time Network), and NSFNET (National Science Foundation Network); the last of these would eventually replace ARPANET as the backbone of the Internet before it was itself superseded by commercial networks.

The term "Internet" was adopted in 1983, at about the same time that TCP/IP came into wide use. In 1983, ARPANET was divided into two parts, MILNET, to be used by military and defense agencies, and a civilian version of ARPANET. The word "Internet" was initially coined as an easy way to refer to the combination of these two networks, to their "internetworking."

The end of ARPANET's days arrived in mid-1982, when its communications protocol, NCP, was turned off for a day, allowing only network sites that had switched to Cerf's TCP/IP language to

communicate. On January 1, 1983, NCP was consigned to history, and TCP/IP began its rise as the universal protocol. The final breakthrough for TCP/IP came in 1985, when it was built into a version of the UNIX operating system. That eventually put it in Sun Microsystems workstations and, Naughton writes, "into the heart of the operating system which drove most of the computers on which the Internet would eventually run." As Cerf would observe, "The history of the Net is the history of protocols."

As both free and commercial online services like Prodigy, FidoNet, Usenet, Gopher, and many others rose, and as NSFNET became the Internet's backbone, ARPANET's importance diminished. The system was finally shut down in 1989 and formally decommissioned in 1990, just two years before Tim Berners-Lee would change everything all over again with the introduction of the World Wide Web.

3.2 Tim Berners-Lee and the Invention of the World Wide Web

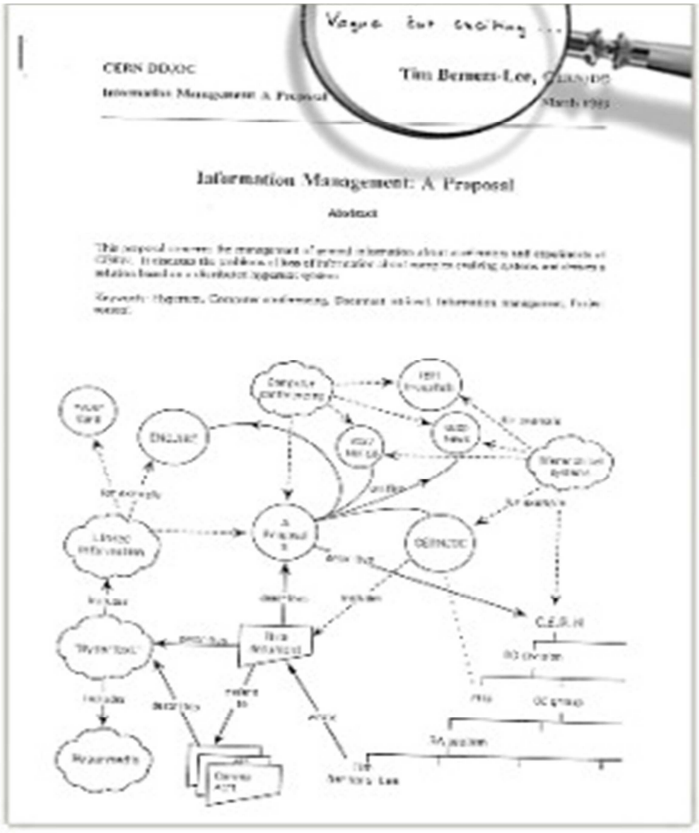
Sir Tim Berners-Lee is a British computer scientist. He was born in London, and his parents were early computer scientists, working on one of the earliest computers.

Growing up, Sir Tim was interested in trains and had a model railway in his bedroom. He recalls:
"I made some electronic gadgets to control the trains. Then I ended up getting more interested in electronics than trains. Later on, when I was in college I made a computer out of an old television set."
After graduating from Oxford University, Berners-Lee became a software engineer at CERN, the large particle physics laboratory near Geneva, Switzerland. Scientists come from all over the world to use its accelerators, but Sir Tim noticed that they were having difficulty sharing information.

"In those days, there was different information on different computers, but you had to log on to different computers to get at it. Also, sometimes you had to learn a different program on each computer. Often it was just easier to go and ask people when they were having coffee…", Tim says.
Tim thought he saw a way to solve this problem – one that he could see could also have much broader applications. Already, millions of computers were being connected together through the fast-developing internet and Berners-Lee realized they could share information by exploiting an emerging technology called hypertext.

In March 1989, Tim laid out his vision for what would become the web in a document called "Information Management: A Proposal". Believe it or not, Tim's initial proposal was not immediately accepted. His boss at

the time, Mike Sendall, noted the words “Vague but exciting” on the cover. The web was never an official CERN project, but Mike managed to give Tim time to work on it in September 1990. He began work using a NeXT computer, one of Steve Jobs’ early products.



Tim’s original proposal. Image: CERN

By October of 1990, Tim had written the three fundamental technologies that remain the foundation of today’s web (and which you may have seen appear on parts of your web browser):

HTML: HyperText Markup Language. The markup (formatting) language for the web.

URI: Uniform Resource Identifier. A kind of “address” that is unique and used to identify each resource on the web. It is also commonly called a URL.

HTTP: Hypertext Transfer Protocol. Allows for the retrieval of linked resources from across the web.

Tim also wrote the first web page editor/browser (“WorldWideWeb.app”) and the first web server (“httpd”). By the end of 1990, the first web page was served on the open internet, and in 1991, people outside of CERN were invited to join this new web community.

As the web began to grow, Tim realized that its true potential would only be unleashed if anyone, anywhere could use it without paying a fee or having to ask for permission.

He explains: "Had the technology been proprietary, and in my total control, it would probably not have taken off. You can't propose that something be a universal space and at the same time keep control of it."
So, Tim and others advocated to ensure that CERN would agree to make the underlying code available on a royalty-free basis, forever. This decision was announced in April 1993 and sparked a global wave of creativity, collaboration, and innovation never seen before. In 2003, the companies developing new web standards committed to a royalty-free policy for their work. In 2014, the year we celebrated the web's 25th birthday, almost two in five people around the world were using it.

Tim moved from CERN to the Massachusetts Institute of Technology in 1994 to found the World Wide Web Consortium (W3C), an international community devoted to developing open web standards. He remains the Director of W3C to this day.

The early web community produced some revolutionary ideas that are now spreading far beyond the technology sector:
Decentralisation: No permission is needed from a central authority to post anything on the web, there is no central controlling node, and so no single point of failure … and no "kill switch"! This also implies freedom from indiscriminate censorship and surveillance.

Non-discrimination: If I pay to connect to the internet with a certain quality of service, and you pay to connect with that or a greater quality of service, then we can both communicate at the same level. This principle of equity is also known as Net Neutrality.

Bottom-up design: Instead of code being written and controlled by a small group of experts, it was developed in full view of everyone, encouraging maximum participation and experimentation.

Universality: For anyone to be able to publish anything on the web, all the computers involved have to speak the same languages to each other, no matter what different hardware people are using; where they live; or what cultural and political beliefs they have. In this way, the web breaks down silos while still allowing diversity to flourish.

Consensus: For universal standards to work, everyone had to agree to use them. Tim and others achieved this consensus by giving everyone a say in creating the standards, through a transparent, participatory process at W3C.

New permutations of these ideas are giving rise to exciting new approaches in fields as diverse as information (Open Data), politics (Open Government), scientific research (Open Access), education, and culture (Free Culture). But to date, we have only scratched the surface of how these principles could change society and politics for the better.

In 2009, Sir Tim co-founded the World Wide Web Foundation with Rosemary Leith. The Web Foundation is fighting for the web we want a web that is safe, empowering, and for everyone.

3.3 Evolution of the web: from static to dynamic content

Web development has undergone a remarkable evolution since its inception, transforming from simple static web pages to highly interactive and dynamic experiences. This journey has been fueled by advancements in technology, changes in user expectations, and the ever-growing demand for richer online experiences.

The Era of Static Web Pages: In the early days of the World Wide Web, websites were predominantly static, consisting of simple HTML pages with limited interactivity. These static websites served primarily as digital brochures, providing users with basic information about businesses, organizations, or individuals. Web developers focused on creating visually appealing layouts and ensuring compatibility across different web browsers.

Introduction of Dynamic Content: The emergence of server-side technologies, such as CGI (Common Gateway Interface) and PHP (Hypertext Preprocessor), revolutionized web development by enabling the generation of dynamic content. With server-side scripting, websites could now retrieve data from databases, handle user input, and personalize content based on user interactions. This shift paved the way for more interactive web experiences, including e-commerce platforms, discussion forums, and content management systems.

Rise of Client-Side Scripting: As internet speeds improved and browser capabilities advanced, client-side scripting languages like JavaScript gained prominence. JavaScript allowed developers to manipulate web page elements dynamically, create interactive features, and enhance user interfaces without requiring round-trips to the server. This led to the development of rich internet applications (RIAs) and AJAX (Asynchronous JavaScript and XML) techniques, which enabled smoother and more responsive user experiences.

The Era of Web Frameworks: With the increasing complexity of web applications, developers turned to web frameworks to streamline

development processes and maintain code consistency. Frameworks like Ruby on Rails, Django, and Laravel provided pre-built components, MVC (Model-View-Controller) architecture, and other tools to accelerate development and ensure scalability. These frameworks empowered developers to focus on building features and functionality rather than reinventing the wheel with each project.

Mobile Responsiveness and Progressive Web Apps (PWAs): The proliferation of smartphones and tablets prompted a shift towards mobile-first web development practices. Websites needed to adapt to various screen sizes and device capabilities to deliver optimal user experiences across different platforms. Responsive web design became the norm, ensuring that websites dynamically adjusted their layout and content based on the viewing device. Additionally, Progressive Web Apps (PWAs) emerged as a hybrid approach, combining the best features of web and mobile applications to deliver fast, reliable, and engaging experiences, even in offline mode.

The Era of Single Page Applications (SPAs) and APIs: Single Page Applications (SPAs) introduced a paradigm shift in web development, offering seamless, fluid user experiences akin to desktop applications. SPAs load content dynamically, updating the page without full page reloads, resulting in faster navigation and smoother interactions. Technologies like React, Angular, and Vue.js empowered developers to build complex SPAs with reusable components and efficient data management. Moreover, the rise of web APIs (Application Programming Interfaces) facilitated seamless integration with third-party services and data sources, enabling developers to leverage a wealth of resources to enrich their applications.

Embracing Modern Technologies: Web development continues to evolve rapidly, driven by advancements in technologies such as WebAssembly, GraphQL, and serverless computing. WebAssembly enables high-performance, language-agnostic code execution in the browser, opening the door to new possibilities for web-based applications, including gaming, multimedia processing, and augmented reality. GraphQL simplifies data fetching and manipulation by providing a flexible and efficient query language for APIs, empowering clients to request precisely the data they need. Serverless computing abstracts away infrastructure management, allowing developers to focus on writing code without worrying about server provisioning or scalability.

The evolution of web development has been characterized by a relentless pursuit of richer, more engaging user experiences. From the humble beginnings of static web pages to the complex, interactive applications of today, developers have continually pushed the boundaries of what is possible on the web. As technology continues to

advance and user expectations evolve, web developers will undoubtedly face new challenges and opportunities, shaping the future of the digital landscape for years to come.

Self-Assessment Exercise(s)

(1) Who is considered the "father of the Internet"?
a) Tim Berners-Lee
b) Vint Cerf
c) Bill Gates
d) Steve Jobs
Answer: b) Vint Cerf

(2) Which organization developed the ARPANET, the precursor to the modern Internet?
a) NASA
b) IBM
c) DARPA
d) CERN
Answer: c) DARPA

(3) In what year was the World Wide Web invented?
a) 1983
b) 1989
c) 1991
d) 1995
Answer: b) 1989

(4) Who invented the World Wide Web?
a) Marc Andreessen
b) Larry Page
c) Tim Berners-Lee
d) Sergey Brin
Answer: c) Tim Berners-Lee

(5) Which protocol is fundamental for the functioning of the Internet by providing unique addresses for devices?
a) HTTP
b) FTP
c) IP
d) SMTP
Answer: c) IP

Conclusion

The history of the Internet and the Web is a remarkable narrative of innovation, collaboration, and societal transformation. Originating from the early experiments in packet switching and ARPANET, the Internet evolved into a global network that connects billions of devices and people. The invention of the World Wide Web by Tim Berners-Lee in 1989 revolutionized how information is shared and accessed, transforming the Internet from a tool for researchers and academics into an indispensable resource for everyday life. This evolution has been marked by significant technological advancements, such as the development of TCP/IP protocols, the creation of user-friendly web browsers, and the rise of dynamic web applications. The Internet and the Web have not only reshaped communication and commerce but also posed new challenges and opportunities for privacy, security, and digital equity. As we continue to navigate this digital landscape, understanding its history helps us appreciate its impact and guides us in addressing the complexities of its future.

4.0 Summary

The history of the Internet began in the late 1960s with the development of ARPANET by the Defense Advanced Research Projects Agency (DARPA). This early network used packet switching to allow multiple computers to communicate on a single network, laying the groundwork for modern Internet protocols. In 1983, the adoption of the TCP/IP protocol suite enabled various disparate networks to interconnect and communicate effectively, marking a significant milestone in the evolution of the Internet. The network continued to grow, supported by academic and governmental bodies, eventually opening up to commercial use and the broader public by the late 1980s and early 1990s.

The World Wide Web, invented by Tim Berners-Lee in 1989, fundamentally transformed the Internet by introducing a system for sharing information through interconnected documents and multimedia. The first web browser, World Wide Web (later renamed Nexus), and the subsequent release of Mosaic in 1993, made the Web accessible to a wider audience. This era saw rapid growth in the number of websites and users, leading to the dot-com boom of the late 1990s. Over time, the Web evolved from static pages to dynamic, interactive platforms, epitomized by Web 2.0 technologies like social media, blogs, and wikis. This transformation has had profound impacts on communication, commerce, and culture, shaping the digital landscape we navigate today.

Understanding this history highlights the technological advancements and societal shifts that have defined our modern era.

5.0 References/Further Readings

Lambert, L., Woodford, C., & Moschovitis, C. J. (2005). The internet: a historical encyclopedia (Vol. 2). Abc-clio.

McCullough, B. (2018). How the internet happened: from Netscape to the iPhone. Liveright Publishing.

Trinkle, D. A., & Merriman, S. A. (2002). The World History Highway: A Guide to Internet Resources. ME Sharpe.

Ducke, I. (2003). Activism and the Internet: Japan's 2001 history-textbook affair. In Japanese cybercultures (pp. 223-239). Routledge.

Trinkle, D. A., Auchter, D., Merriman, S. A., & Larson, T. E. (2016). The history highway: A 21st-century guide to internet resources. Routledge.

Unit 2 Understanding Web Browsers

Contents

1.0 Introduction

A web browser takes you anywhere on the internet. It retrieves information from other parts of the web and displays it on your desktop or mobile device. The information is transferred using the Hypertext Transfer Protocol, which defines how text, images, and video are transmitted on the web. This information needs to be shared and displayed in a consistent format so that people using any browser, anywhere in the world can see the information. Sadly, not all browser makers choose to interpret the format in the same way. For users, this means that a website can look and function differently. Creating consistency between browsers, so that any user can enjoy the internet, regardless of the browser they choose, is called web standards.

When the web browser fetches data from an internet-connected server, it uses a piece of software called a rendering engine to translate that data into text and images. This data is written in Hypertext Markup Language (HTML) and web browsers read this code to create what we see, hear, and experience on the internet. Hyperlinks allow users to follow a path to other pages or sites on the web. Every webpage, image and video has its own unique Uniform Resource Locator (URL), which is also known as a web address. When a browser visits a server for data, the web address tells the browser where to look for each item that is described in the HTML, which then tells the browser where it goes on the web page.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Understand the web browsers.
- Identify various types of web browsers.

3.0 Main Content

3.1 Introduction to web browsers and their role in accessing the web

When we need any kind of information most of the time we get help from the Internet, and we get information. The Internet provides us with useful information easily. We use mobile phones, computers, and tablets. We search for a lot of things in our daily lives, so we get information about all over the world, but we cannot get information by just only getting connected to the Internet. We need a platform where we can search for our questions. The platform that provides such kinds of services is called a web browser, without a web browser internet will not be able to provide information.

The web browser is an application software to explore www (World Wide Web). It provides an interface between the server and the client and it requests to the server for web documents and services. It works as a compiler to render HTML which is used to design a webpage. Whenever we search for anything on the internet, the browser loads a web page written in HTML, including text, links, images, and other items such as style sheets and JavaScript functions. Google Chrome, Microsoft Edge, Mozilla Firefox, and Safari are examples of web browsers.

The first web browser World Wide Web was invented in the year of 1990 by Tim Berners-Lee. Later, it becomes Nexus. In the year of 1993, a new browser Mosaic was invented by Mark Andreessen and their team. It was the first browser to display text and images at a time on the device screen. He also invents another browser Netscape in 1994. Next year Microsoft launched a web browser Internet Explorer which was already installed in the Windows operating system. After this many browsers were invented with various features like Mozilla Firefox, Google Chrome, Safari, Opera, etc.

You might wonder that if every web browser provides an interface to the user and allows them to connect with the web, then why are there so many web browsers in the first place? If we look at the history of web

browsers, we can tell a lot about their special features and developing techniques.

Before web browsers came into existence, computers were more like boxes that used command-line interfaces to perform some tasks. The development of web browsers revolutionized the entire way of interacting with a computer.

In 1990, a computer scientist Tim Berners-Lee, developed the first ever browser named “World Wide Web”, at CERN (A European organization for nuclear research). It was indeed an extraordinary development as it was the only web browser present at that time that provided a user-friendly interface.

In 1993, everyone started becoming familiar with the concept of web browsers and was curious to develop the same. This led to the invention of the “Mosaic” by NCSA (National Centre for Supercomputing Applications) at the University of Illinois. It gained massive popularity as it was the first graphical browser that demonstrated the use of multimedia (images) on the web.

In 1994, another browser called “Netscape”, founded by Andreessen turned out to be a success. It was the first web browser to be made public and gained immense popularity.

In 1995, the world witnessed a race to develop better web browsers with attractive versions. It was this year when Microsoft developed the “Internet Explorer” and released it resulting in so-called “browser wars”. The war between Internet Explorer and Netscape continued for a while. In 1996, Opera Software released its web browser” Opera”.

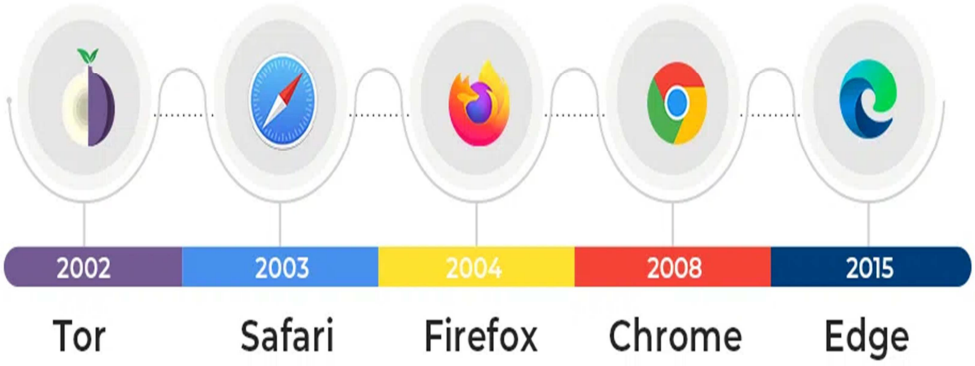
In 1998, Netscape made its code open source which led to the birth of “Mozilla”. It later evolved into the Mozilla Firefox web browser, which eventually gained user’s attraction.

In 2003, Apple launched “Safari” a web browser particularly made for devices with the Macintosh operating system. Later it was available for Windows users too and even had a mobile version.

In 2008, the “Google Chrome” web browser, founded by Google, came into existence and started taking over the entire market of web browsers and ruling them. It attracted users all across the globe because of its high-speed rendering, improved web performance, and well-designed interface. To date, it has maintained its position as the most used web browser.

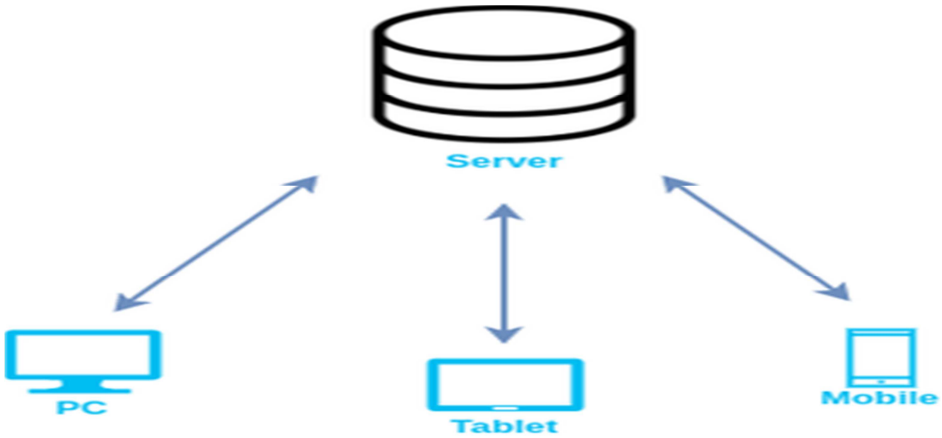
Later in 2015, Internet Explorer was renamed as “Microsoft Edge“. In today’s time, Microsoft Edge comes preinstalled on many Windows devices. Also, several other browsers like Brave and Vivaldi came into the game and are used by some users for their unique features.

Timeline of Browsers:
Given below is the timeline of browsers found to date:



3.2 How does a Web Browser Work?

A web browser helps us find information anywhere on the internet. It is installed on the client computer and requests information from the web server such a type of working model is called a client-server model.



Client-server model
The browser receives information through HTTP protocol. In which transmission of data is defined. When the browser receives data from the server, it is rendered in HTML to user-readable form, and, information is displayed on the device screen.

Website Cookies
When we visit any website over the internet our web browser stores information about us in small files called cookies. Cookies are designed to remember stateful information about our browsing history. Some more cookies are used to remember about us like our interests, our

browsing patterns, etc. Websites show us ads based on our interests using cookies.

An Overview of Web Browser Functionality:

Web browsers are software programs that provide an interface or medium for users to connect with the web. It receives requests sent by users, fetches data from web servers, and then displays the result to the user.

The user always searches in the form of a URL (for example, www.noun.edu.ng) and sends a request to the browser.

Then the browser converts the URL into a numeric value, which is called an IP address, using DNS, or Domain Name System.

Through the IP address, web browsers generate a request, which is sent to the web servers.

The data retrieved from the server (which is in the form of code) is parsed, and then when the entire document object model is created, it is presented to users in the form of web pages.

Advantages of using Web Browsers:

A web browser's main role is to provide an interface or medium through which users can search on the web and access various websites.

Users can visit different websites by just entering the URL of the website on the web browser. Thus, it makes surfing on internet easier and quicker.

Web browsers also ensures secure and safe access to websites and prevents users' device from any malicious software. They take special care of users' privacy by data encryption.

Multimedia (images and videos) is also supported by web browsers. It displays multimedia elements using CSS and JavaScript.

Several web browsers provide features to sync devices, thus making it easier for the user to access their personalized settings on multiple devices.

In current times, users demand a faster, more efficient, and more secure browsing experience. Keeping this in mind, various features are being constantly added, and the web browsers are updated from time to time. Every browser has basic features like an address bar, navigation buttons, a home button, a refresh, history, downloads, etc.

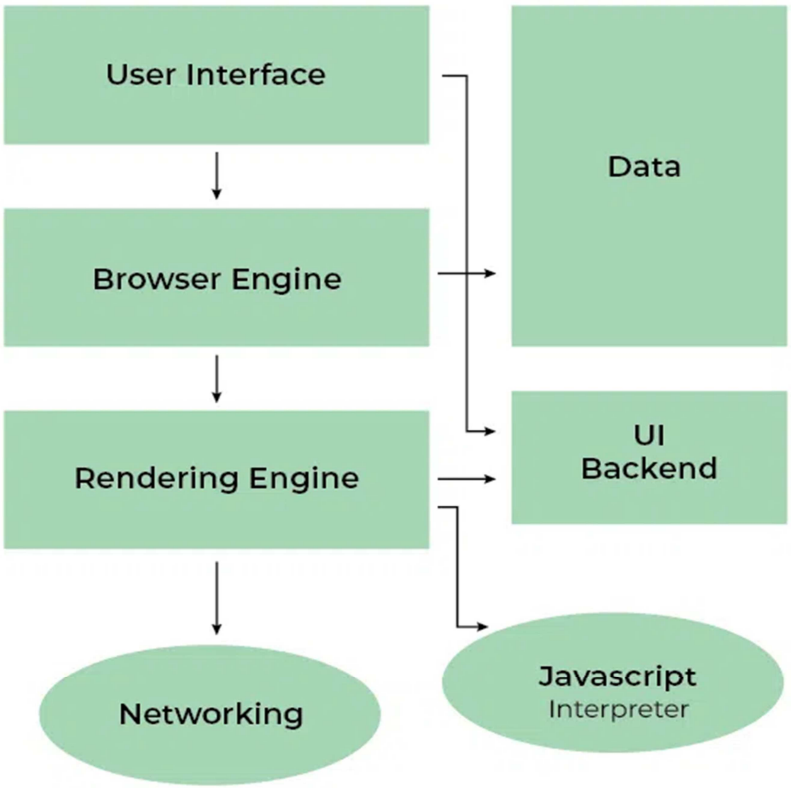
Some more features of web browsers present in today's time are:
Web browsers keep on making updates, keeping one of the most important things in mind, which is a "user-friendly experience". For example, Chrome, being the most used web browser, offers attractive features and provides users with an easy and quick environment to work in.

As technology is increasing in modern times, so is the threat of cybercrime or virus attacks. Web browsers now provide a secure setup for users, taking special care of their privacy.

Web browsers also offer a wide range of extensions and add-ons to improve the functioning of web browsers and make them more efficient. Special features like allowing devices to sync, thus preventing loss of data, and adding voice search, screen reading, etc. have made it possible for every person to benefit from web browsers.

3.4 Web Browser Architecture

Browser architecture is designed to provide a faster, more secure, and more feature-rich platform that helps users interact easily with the internet. The browser architecture is broadly divided into seven parts.



Browser Architecture

The user interface of a browser is designed such that it allows personalization, as every individual has different interests. This personalization is achieved by providing basic features like groups, collections, bookmarks, and themes. Each browser can have a different user interface and features.

Browser Engine: The browser engine is responsible for coordinating web content that is fetched from the server and user interactions. It keeps a note of which button is clicked, which URL is asked to parse, and how the web content will be processed and displayed on the browser.

Rendering Engine: The rendering engine, on the other hand, interprets and renders web content. In most browsers, both the browser engine and the rendering engine work together to provide better results to the user.

Networking Layer: This layer handles the communication part. When the user enters or clicks on a URL, the network layer initiates an HTTP request to the webserver to load the requested web page. It also manages fetching resources from HTML files, images, stylesheets, and more. Have you seen those cookie notifications while searching for information on the internet? Mostly, the network layer works behind the scenes for those cookies and cache.

JavaScript Engine: The JavaScript Engine is the core component of browser architecture, with the ability to manipulate web content and introduce dynamic behavior in web pages.

Data Storage: A large part of the browser goes into storing various types of data, which include not only user preferences, browsing history, passwords, and other regular data updates as well (address, name, and contact).

UI backend: The UI backend provides dynamic and interactive behavior on the web page and enhances the overall functionality and performance of the browser.

Self-Assessment Exercise(s)

(1) Which of the following is NOT a popular web browser?
a) Google Chrome
b) Mozilla Firefox
c) Adobe Photoshop
d) Apple Safari
Answer: c) Adobe Photoshop

(2) What is the primary function of a web browser?
a) To compile code
b) To access and display web pages
c) To edit images
d) To manage databases
Answer: b) To access and display web pages

(3) Who is credited with inventing the World Wide Web?
a) Bill Gates
b) Steve Jobs
c) Tim Berners-Lee
d) Mark Zuckerberg
Answer: c) Tim Berners-Lee

(4) Which of the following is a feature commonly found in web browsers?
a) Spreadsheet editing
b) Rendering HTML and CSS
c) Video Production
d) Compiling Java code
Answer: b) Rendering HTML and CSS

(5) What does the term "rendering" refer to in the context of web browsers?
a) Storing data on a server
b) Displaying the visual layout of a web page
c) Encrypting user data
d) Sending emails
Answer: b) Displaying the visual layout of a web page

 **Conclusion**

The magic of web browsing doesn't end with its convenience. Understanding the intricate workings behind the scenes, from rendering engines to JavaScript manipulation, adds a layer of appreciation for the complex symphony that delivers information to your screen at lightning speed. Whether you're a tech enthusiast or simply curious about how things work, delve deeper into browser architecture to unlock a new level of understanding about the tool that connects you to the vast world of the internet.

4.0 Summary

Understanding Web Browsers" is an essential module that delves into the function and importance of web browsers, the primary tools for accessing and navigating the World Wide Web. This unit explains how web browsers work, detailing their role in interpreting and displaying web content by processing HTML, CSS, and JavaScript. Students learn about the architecture of browsers, including the rendering engine, JavaScript engine, and networking components, which collaborate to provide a seamless browsing experience. In addition to the technical aspects, the unit covers practical usage and features of web browsers that enhance user experience.

5.0 References/Further Readings

Brusilovsky, P., Schwarz, E., & Weber, G. (1996, October). A tool for developing adaptive electronic textbooks on WWW. Web site: http://aace. virginia. edu/aace/conf/webnet/html/151/151. htm.

Ducke, I. (2003). Activism and the Internet: Japan’s 2001 history-textbook affair. In Japanese cybercultures (pp. 223-239). Routledge.

Trinkle, D. A., Auchter, D., Merriman, S. A., & Larson, T. E. (2016). The history highway: A 21st-century guide to internet resources. Routledge.

Unit 3 Internet Services, Communication, and Protocol

Contents

1.0 Introduction

This unit delves into the foundational aspects and intricate mechanisms that power the global network of interconnected computers known as the Internet. This unit aims to provide a comprehensive understanding of the various internet services that facilitate communication, information sharing, and digital transactions in today's world. By exploring the architecture and functionalities of essential services like the World Wide Web, email, file transfer, and streaming, learners will gain insights into how these services operate seamlessly and efficiently to meet the demands of billions of users worldwide. Central to this unit is the exploration of communication protocols, the rules, and conventions that govern the exchange of data over the Internet. These protocols ensure reliable, secure, and standardized communication between diverse systems and devices. The unit will cover key protocols such as HTTP, HTTPS, FTP, SMTP, and TCP/IP, explaining their roles, functionalities, and significance in maintaining the internet's integrity and performance. By the end of this unit, students will have a solid grasp of how internet services and communication protocols work in unison to create a cohesive and dynamic digital ecosystem.

2.0 Intended Learning Outcomes (ILOs)

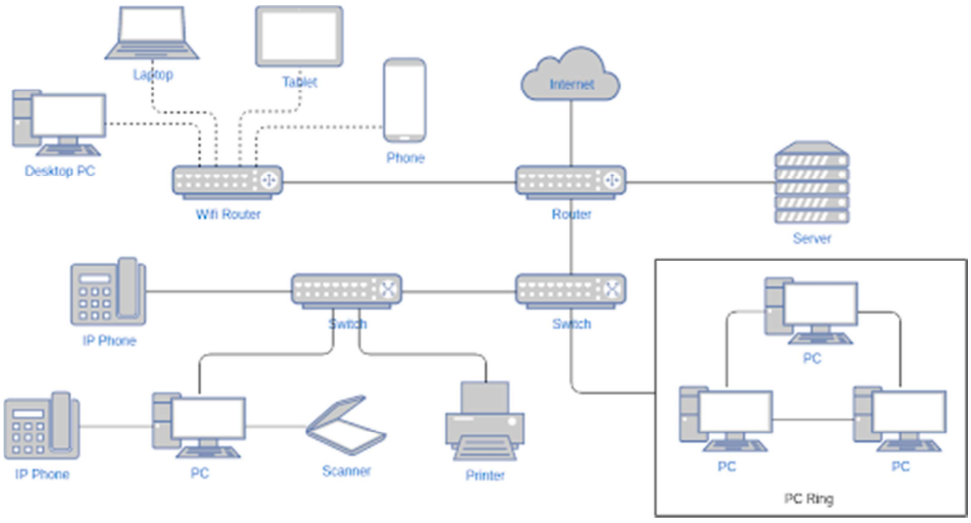
By the end of this unit, you will be able to:

- Understand Internet.
- Identify various services the Internet provides for the users.
- Explain the Internet communication channels
- Explain some common Internet Protocols

3.0 Main Content

3.1 The Internet Services

The Internet is defined as a global network of linked computers, servers, phones, and smart appliances that communicate with each other using the transmission control protocol (TCP) standard to enable the fast exchange of information and files, along with other types of services. The Internet is a global network of interconnected computers, servers, phones, and smart appliances that communicate with each other using the TCP standard to enable a fast exchange of information and files, along with other types of services.



The internet is a global hub of computer networks, a network of connections wherein users at any workstation may, with authorization, receive data from every other system (and often interact with users working on other computers).

Internet infrastructure comprises optical fiber data transmission cables or copper wires, as well as numerous additional networking infrastructures, such as local area networks (LAN), wide area networks (WAN), metropolitan area networks (MAN), etc. Sometimes wireless services such as 4G and 5G or WiFi necessitate similar physical cable installations for internet access.

Internet Corporation for Assigned Names and Numbers (ICANN) in the United States controls the Internet and its associated technologies, such as IP addresses.

How was the Internet developed?
The internet was first envisioned in the form of ARPANET by the Advanced Research Projects Agency (ARPA) of the U.S. government in 1969. The initial goal was to create a network that would enable users of a research computer at one institution to "communicate" with research computers at another institution. Since communications can be sent or diverted across several directions, ARPANet could continue to operate even if a military strike or any other calamity damages portions of the network.

ARPANET used the new packet-switching technology to create low-cost, interactive interactions between computers, which generally communicate in short data bursts. Packet switching broke down large transmissions (or portions of computer data) into smaller, more manageable parts (called packets) that could travel independently across any accessible circuit to the destination where they were reassembled. Consequently, unlike conventional voice services, packet switching doesn't require a separate dedicated connection between a pair of users.
In the 1970s, corporate packet networks were launched, although their primary purpose was to enable efficient access to distant computers through specialized terminals. They replaced expensive long-distance modem connections with "virtual" lines via packet networks.

Today, the Internet is a globally accessible, collaborative, and self-sustaining public resource available to tens of millions of individuals. Countless people utilize it as their primary source of data consumption, spurring the development and expansion of their own community through social networking and content exchange. However, private versions of the internet do exist, which are primarily used by large organizations for secure and regulated information exchange.

Key features of the Internet
The internet is a vast, interconnected network of computers and other network-enabled devices, which is:

Globally available: The internet is an international service with universal access. People living in isolated areas of an archipelago or even in the depths of Africa can now access the internet.

Easy to use: The software used to connect to the internet (web browser) is user-friendly and easy to understand. It's also relatively easy to create. Compatible with other types of media: The internet provides a high level of engagement with photos and videos, among other media.

Affordable: Internet service development, as well as maintenance costs, are modest.

Flexible: Internet-based communication is highly adaptable. It supports text, audio, and video communication. These services are available at both individual and organizational levels.

How Does the Internet Work?

The internet delivers different types of information and media across networked devices. It operates using an internet protocol (IP) and a transport control protocol (TCP) packet routing network. Whenever you visit a website, your computer or mobile device requests the server using such protocols.

A server is where web pages are stored, and it functions similarly to the hard drive of a computer, except with far greater processing power. The server accesses the web page and delivers the right information to your computer whenever the request arrives. This is broadly the end-to-end user experience. Let us now look at the more technical details of how the internet works.

1. **Connecting computers:** The basic foundation of the internet is an interconnected network of computers. When two computers interact, they must be physically (often via an Ethernet connection) or wirelessly connected (via Wi-Fi or Bluetooth). All modern systems can support any of these connections to establish a core network.
2. **Scaling computer networks:** The computer network, as described above, is not restricted to two PCs. One can link several computers. However, as you expand, it may get more complex. Every machine on a network is connected to a tiny computing device known as a router to address this problem. This router's only function is to operate as a signaler. It ensures that a message transmitted from a particular computer reaches its intended recipient. With the addition of a router, a system of 10 computers needs merely ten wires instead of $10 \times 10 = 100$ connections.
3. **Enabling infinite scaling:** Let us now discuss interconnecting hundreds of thousands to billions of machines. A single router cannot scale to that extent; nonetheless, a router is an independently programmable computer unit. This implies that two or more routers may be connected, enabling infinite scaling.
4. **Utilizing ubiquitous public infrastructure via a modem:** By now, we have constructed a network identical to the internet, although it is only intended for individual use and cannot connect with the outside world. This is where public infrastructure comes in. The telephone system links an office to everyone worldwide, making it the ideal wiring configuration for the Internet. A modem is necessary for connecting networks to the telephone

system. This modem converts data from a network into data that can be managed by the telephony architecture and vice versa.

5. **Sending messages from one network to another:** The following step is to transmit the information from your network to the target network. To accomplish this, the network must establish a connection with an internet service provider (ISP). An ISP is a service that administers specified routers that are interconnected and also have access to the routers of other ISPs. Therefore, the data from the host network is delivered to the target network via the web of ISP networks.

   To deliver a message to a system, it is important to identify which computer it should be sent to. Therefore, every machine connected to a network has a unique identifying address known as an "IP address" (here, IP refers to internet protocol). It is an address consisting of four integers separated by periods, such as 192.168.2.10. There are several versions of IP; currently, we are in IPv4 and IPv6 iterations, depending on the region.
6. **Assigning domain name to IP addresses:** IP addresses are intended for computers, but in an infinitely extensible internet, it would be difficult for people to keep count of an ever-growing number of addresses. To simplify matters, one may designate an IP address with a domain name, a human-readable name. Google.com is an excellent example of this — the domain name is used in conjunction with the IP address 142.250.190.78. Therefore, typing the domain name is the simplest way to access a computer online.
7. **Connected the internet to the web:** The internet is a network architecture that enables millions of machines to communicate with one another. Several of these machines (web servers) can feed web browsers intelligible messages. The web is an application constructed on top of the internet's infrastructure. It is important to note that additional services, like email, have been developed on top of the internet.
8. **Connecting the internet to a private intranet or extranet:**Intranets are personal and bespoke networks confined to an organization's members. They offer participants a secure gateway to access shared information, collaborate, and communicate.
   Extranets are quite similar to intranets, except that they enable collaboration and sharing with other businesses. Typically, they are employed to safely and confidentially transmit information to customers and other enterprise stakeholders. Frequently, their functions resemble those of an intranet: file and information sharing, collaboration tools, message boards, etc.

Intranets and extranets operate on the same infrastructure and adhere to the same protocols as the internet.

How does the web work?
When we discuss the internet in common parlance, we typically refer to the web – although the two terms are not interchangeable. If the internet can be understood as a network of highways, then the web will be the network of restaurants, toll booths, gas stations, etc., built along it. The main job of the internet is to access the web. However, it can perform other tasks like supporting cloud storage on computers, keeping the software as a service (SaaS) apps online, automatically updating the computer's time, etc.

On the other hand, the web comprises multiple computers connected to the internet called clients and servers.

Clients are internet-connected devices of a web user (such as a computer linked to Wi-Fi or a mobile phone) and the online-accessing software installed on such systems (generally a web browser).

Servers store websites, applications, and their associated data and activities. When a client device requests access to a website, a replica of the webpage is received from the server to the client's computer. The webpage is then exhibited in the client's web browser.

When a user inputs a domain name or uniform resource locator (URL) in the browser, the domain name system (DNS server) is contacted to get the actual IP address of the website's server.

The browser then transmits an HTTP or HTTPS request message back to the server, asking the server to transmit a copy of the web page to the client. This message and all other data transferred between the client and server are sent via the TCP/IP protocol across your internet connection.
If the server authorizes the client's request, it returns a "200 OK" status code. The server then begins transmitting the site's contents to the client as a sequence of data packets. The browser constructs an entire web page from the packets and starts displaying it. This request, response, and information exchange happens via the internet infrastructure.

3.2 Types of Internet Services

As mentioned earlier, the internet can enable various services, not just web access. Some of the key types of Internet services are:

1. **Communication services:** To exchange data/information among individuals or organizations, the Internet enables communication services. This mainly includes VoIP and video conferencing.

Voice over Internet protocol (VoIP) enables users to place voice calls over the Internet compared to a conventional (or analog) phone connection. Other VoIP services allow you to contact anybody with a mobile number, encompassing long-distance, cellular, and even local/international connections.

Video conferencing technology enables two or more individuals in separate locations to connect visually and in real-time. It includes persons in different places using video-enabled devices and broadcasting real-time speech, video, texts, and slideshows via the internet.

Other communication services based on the Internet include email, internet relay chat (IRC), and list server (LISTSERV) used for asynchronous text communication, instant messaging, and group announcements, respectively.

2. **File transfer services:** We utilize file transfer to exchange, transmit, or send a document or logical data item among many individuals or computers, both locally and remotely. Data files may comprise documents, videos, photos, text, or PDFs. They may be shared via internet downloading and uploading. File transfer protocol (FTP) is one of the most common internet protocols used for this purpose.
3. **Directory services:** A directory service is a collection of software that maintains information about the organization, its customers, or both. Directory services are responsible for mapping network resource names to network addresses. It offers administrators and users transparent access to all network computers, printers, servers, and other devices. It is also an important backend service provider for and by the internet.

   Domain number system (DNS) and lightweight directory access protocol (LDAP) are the most commonly used directory services. A DNS server stores a map of computer hostnames and other domain names to IP addresses. LDAP is a collection of open protocols to obtain centralized network access to stored data. It is also a mechanism for cross-platform authentication.
4. **E-commerce and online transactions:** E-commerce allows the customer to purchase a service or product directly from the vendor, at any time or anywhere on the planet. When IBM started offering hardware and software for computers over the Internet, it was one of the first instances of e-commerce. Since then, this service has grown in use tremendously. E-commerce uses the web to enable financial exchanges so that data packets can translate into their real-world monetary equivalents.

5. **Services for network management:** Network management services are some of the most critical and valuable Internet services for IT administrators. They assist in avoiding, monitoring, diagnosing, and resolving network-related issues. Two services are mainly used for this purpose – ping and traceroute.

   The ping utility checks the host machine's availability and the time required to react to any internet control message protocol (ICMP) transmissions. It guarantees that all requests issued by a computer reach the web server without packet loss. In the meantime, the traceroute identifies and displays all potential paths from query to response, as well as the turnaround time for each route.
6. **Time services:** Greenwich Mean Time (GMT) or Coordinated Universal Time synchronizes computer clocks (UTC). Network time protocol (NTP) is an established internet time service that syncs and adjusts the computer clock accurately to all these standards. All Windows time variants released after Windows 2000 synchronize with an NTP server. NTPsec is primarily a secured version of NTP.
7. **Search engine services on the web:** When users search for a web page through a search engine rather than the domain name, the search engine examines the web crawler's index of all pages. It will study the search phrase and compare it to the database, including how often the search terms appear on a webpage, where they appear on the site, whether they appear together, etc. It analyzes this information to determine which websites best fit your search query.
   The results are then shown in order, with those that best fit the search keyword appearing initially. It is important to note that search engines can accept funds from commercial entities to prioritize their websites in the results of a particular query. This is an advert, and the search engine results will be labeled as such.
   Automatic Network Address Configuration: Automatic Network Addressing assigns a unique IP address to every system in a network. A DHCP Server is a network server that is used to assign IP addresses, gateways, and other network information to client devices. It uses Dynamic Host Configuration Protocol as a common protocol to reply to broadcast inquiries from clients.
8. **Network Management Services:** Network management services are another essential internet service that is beneficial to network administrators. Network management services aid in the prevention, analysis, diagnosis, and resolution of connection problems. The two commands related to this are:

ping: The ping command is a Command Prompt command that is used to see if a source can communicate with a specific destination & get all the possible paths between them.
traceroute: To find the path between two connections, use the traceroute command.

9. **Time Services:** Using facilities included in the operating system, you may set your computer clock via the Internet. Some services are :
   Network Time Protocol (NTP): It is a widely used internet time service that allows you to accurately synchronize and adjust your computer clock.
   The Simple Network Time Protocol (SNTP): It is a time-keeping protocol that is used to synchronize network hardware. When a full implementation of NTP is not required, then this simplified form of NTP is typically utilized.
10. **Usenet:** The 'User's Network' is also known as Usenet. It is a network of online discussion groups. It's one of the first networks where users may upload files to news servers and others can view them.
11. **News Group:** It is a lively Online Discussion Forum that is easily accessible via Usenet. Each newsgroup contains conversations on a certain topic, as indicated by the newsgroup name. Users can use newsreader software to browse and follow the newsgroup as well as comment on the posts. A newsgroup is a debate about a certain topic made up of notes posted to a central Internet site and distributed over Usenet, a global network of news discussion groups. It uses Network News Transfer Protocol (NNTP).

3.3 Internet Communication

The exchange of information, data, or messages over the Internet between two or more people or devices is referred to as Internet communication.

It makes use of a variety of communication technologies, including email, instant messaging, social media, video conferencing, and (Voice over Internet Protocol) VoIP Services. Internet communication allows people to connect with each other from anywhere in the world and at any time, as long as they have Internet access. It has transformed the way people communicate, making it faster, easier, and more convenient than ever before. Internet communication is also important in business and commerce, allowing businesses to communicate in real-time with their customers, suppliers, and employees. It has created new opportunities for remote work, collaboration, and international trade.

We all will accept that the Internet is a revolutionary invention for humanity. It not only offers an advanced methodology of communication but also benefits us in multiple ways. Today we will talk about internet communication and how it has changed the way we communicate. The Internet gives us wings to access information within seconds and connects the world. It acts as a communication bridge between two entities. With more than 4.95 billion users internet has become the perfect solution for audio and video communication.

In simple language, we can understand internal communication as a method of connecting and talking to people using the Internet instead of services offered by telecommunication service providers like phones, calls, and text messaging. From App to App calling services to virtual telephony solutions, the Internet enables people to connect with anyone globally within a few seconds. Internet communication is quite a reliable alternative to traditional phone calling services. Web communication services are relatively cheaper than phone lines and are a reliable source of virtual interaction that has a vast range of benefits for its users.

Internet communication enables you to communicate with people over the web. The communication process takes place in any form, such as messages, voice, video calls, etc. The biggest reason people are shifting towards this technology for communication is cost savings. You will find several applications over the web that allow app-to-app, click-to-call, and app-to-phone call services without imposing any calling charge for web interaction facility. WhatsApp, Skype, Google Meet, and Messenger are some of the most popular applications that allow users easier access to communication services. Internet communication services not only benefit users but also help businesses to communicate with overseas customers at the most cost-efficient price. Corporations have agreed that they depend on the Internet for smoother functioning of their services.

The Internet offers many advanced options that allow users to have seamless two-way communication. The use of the internet in communication enables users to connect and interact with others without facing any difficulties. It has become a significant mode of connectivity nowadays which benefits the users as well as the businesses in many ways. Let’s find out some of the most influential and prominent modes used for establishing quick and hassle-free communication over the web.

Various forms of Internet communication

1. Instant Messaging

   Instant messaging (IM) is a popular form of Internet communication that enables users to exchange text-based messages, emoji, and sometimes voice or video calls in real time.

It has become a preferred communication tool for both personal and commercial purposes, allowing millions of users to connect with each other instantly.

In the business world, IM has gained significant popularity as a means of communication between employees, and it has also been used to reach out to clients efficiently. It allows for quick and effective passing of information, leading to improved productivity and enhanced customer service. Instant messaging can be accessed through various platforms, including messaging apps, social media, and web-based applications, making it easily accessible and available for use.

IM's popularity is due to its simplicity and ease of use. Users can instantly send and receive messages, receive notifications, and access other features such as file sharing, group chats, and video calls. It has revolutionized the way people communicate, offering an efficient and effective means of communication, without the need for phone calls or email.

Instant messaging is a highly used online chat technology that allows users to communicate with others in real-time via an internet-based chat room.

These systems are specially designed communication tools that enable users to have an uninterrupted communication facility when a user logs into a dedicated instant messaging system. Once he starts his login session, he notifies other users about his presence. Instant messaging tools allow users to chat with each other synchronously. The best thing about these communication tools is, they not only save time but also allow users to have communication facilities without paying a single penny. All they have to pay for internet connection charges and they will be able to communicate with others.

Instant messaging is quite beneficial in several aspects. With the help of these messaging tools, employees can connect with their managers and colleagues to establish communication remotely. The messaging tool eliminates the need to place a call to access information and helps you to have a much easier communication interface for instant information sharing. Yahoo Messenger, Google Messenger, MSN, and Messenger are a few popular instant messaging tools that help users to communicate irrespective of their geographical locations.

2. VOIP and Internet Telephony System

An Internet Telephony System (ITS) is a system that uses VOIP technology to enable communication over the Internet. VOIP is a technology that allows voice communications to take place over the Internet rather than traditional telephone networks that rely on circuit-switched connections. Voice signals are broken down into digital data packets that can be transmitted over the internet and then reassembled at the receiving end with VOIP solutions.

It can include hardware like IP phones, softphones, and Cloud-Based PBX Systems, as well as software applications for voice and video calling. Businesses can benefit from modern cloud telephony features such as call queuing, intelligent call routing, conversational AI, Multi-Level IVR, and conferencing by partnering with a VoIP service provider. Furthermore, VoIP phone solutions have several advantages over traditional phone systems, including lower costs, greater flexibility, and the ability to integrate with other communication tools such as CRM and other tools.

With VoIP technology, businesses can streamline communication processes and save on costs, while also taking advantage of Modern Features to enhance customer engagement and satisfaction. The VoIP phone industry is experiencing rapid growth, as evidenced by the market's impressive valuation of $40 billion in 2022. Experts predict that the industry will continue to expand, with a projected compound annual growth rate (CAGR) of 10% from 2023 to 2032.

Internet telephone services utilize the Internet as a bridge that helps them to route telephonic calls to designated numbers instead of traditional phone lines. The working functionality of VoIP is easily understandable, and it sends voice packets using IP instead of PSPN (public switch telephone network). Once the voice packets reach the pre-decided destinations, it again gets converted into voice data for the receivers.

Internet phone systems are widely being used by users and businesses that need to communicate globally. They are cost-effective and allow you to call globally without imposing high charges.

3. Email

Email, short for electronic mail, is a type of digital communication that allows individuals and businesses to send and receive messages via the Internet. Email messages can include text, images, documents, and other file types, and they

can be sent to one or more recipients. An email has several advantages for both personal and business use. Email is a convenient way to stay in touch with family and friends, share photos and documents, and organize events and schedules for personal use.

Email can be a valuable tool for businesses to communicate with their employees, customers, and partners. Businesses can use it to send important documents, invoices, and other files, as well as customer support and marketing messages. Furthermore, when compared to traditional mail, email is a more cost-effective and environmentally friendly communication method. It allows you to store and organize messages in a digital format, making it simple to search for and refer to previous conversations.

As per a recent survey, you will be amazed to know that the number of active email users will touch 4.3 billion by 2023. It is one of the most effective modes of communication and an advanced way of exchanging media files. Users can have the benefits of these communication facilities without paying any charges. Email is considered one of the most reliable and secure communication channels today, which is why businesses prefer to communicate over email.

Users can create their email ID through web portals like Microsoft Outlook, Gmail, and Yahoo. You only need an active email ID if you want to communicate through the same. Once your ID is activated, you can send messages to the recipient's email address. It is a widely used communication channel by business sectors. The fact is that Emails have become a professional way of communication. 62.86% of businesses prefer email as their prime mode of official communication.

To communicate with someone through Email users need to follow some set pattern of guidelines and steps. They need to mention the recipient's mail address, subject, and text body before sending an email. Emails allow you to go paperless and enable users to send and receive digitalized documents over the channel. With the help of electronic mail, users can connect with multiple people simultaneously and communicate without facing any issues.

4. Video Conferencing

Video conferencing is an essential method of Internet communication for both personal and business purposes. Video conferencing can be used in personal contexts to stay in touch with friends and family who live far away, providing a more

personal and engaging experience than other forms of internet communication. Furthermore, video conferencing can be a valuable tool for remote work situations, allowing people who work from home or in different locations to communicate and collaborate in real-time with colleagues and clients.

Video conferencing can be used for a variety of purposes in the business world. It allows remote teams to collaborate in real-time, share documents and ideas, and hold meetings without having to travel. This can save time and money while also improving communication efficiency. Sales calls, customer service interactions, and virtual events such as webinars and conferences can all benefit from video conferencing. Furthermore, video conferencing can enable businesses to reach a global audience because it enables communication with people in different parts of the world without the need for travel.

5. Social Networking Platforms

Social media is one of the most popular Internet communication methods used by individuals and businesses around the world on a daily basis. Users can connect and interact virtually through these platforms, which provide tools for creating and sharing content, communicating with friends, family, and colleagues, and connecting with others who share similar interests. Facebook, Twitter, Instagram, LinkedIn, and WhatsApp are some of the most popular social networking platforms, each catering to different audiences and use cases.

WhatsApp, for example, allows users to communicate for free through messaging, audio, and video, whereas LinkedIn focuses on professional networking, allowing users to connect with colleagues, search for jobs, and showcase their skills and experience. Social networking platforms have become an essential part of modern communication, influencing how individuals and businesses interact with one another and making communication with friends and customers faster and easier. Businesses can increase brand awareness by using social media platforms, which allow them to interact with customers and provide useful feedback, improving consumer retention. Additionally, social media portals can be used for advertising, promotional campaigns, and market analysis.

Social media is an integral part of our lives and is considered one of the most popular internet communication methods today. You all will be aware of the social media platforms like Facebook, Instagram, Twitter, WhatsApp, etc. These channels have gained immense popularity among every age group.

The social media platform helps users instantaneously communicate with others and offers several features that enhance user experience while communicating. Social media channels are not just limited to chat room facilities. Users are allowed to access multiple features while communicating. They can make video and audio calls along with a chat room facility. Also, they can share various documents and media files over web channels.

3.4 Internet Protocol

Transmission Control Protocol/Internet Protocol (TCP/IP)
The TCP/IP suite is the core set of protocols governing the Internet and many private networks. Developed in the 1970s and early 1980s under the auspices of DARPA (Defense Advanced Research Projects Agency), TCP/IP was designed to facilitate robust and flexible communication across diverse and interconnected networks. The suite is named after its two main protocols: Transmission Control Protocol (TCP) and Internet Protocol (IP).

Key Components
Transmission Control Protocol (TCP): TCP is responsible for ensuring reliable data transmission between devices. It handles the breakdown of large data messages into smaller packets, which are then sent over the network. At the receiving end, TCP reassembles these packets into the original message. TCP ensures data integrity and correct ordering through error detection and correction mechanisms, retransmitting lost packets and acknowledging received packets.

Internet Protocol (IP): IP handles addressing and routing. Each device on a network is assigned an IP address, which is used to identify it uniquely. IP determines the best path for data to travel from the source to the destination, navigating through various networks and routers.

Functionality and Process
TCP/IP operates at multiple layers, each with specific functions:
Application Layer: This layer includes protocols like HTTP, FTP, and SMTP, which are used by applications to communicate over the network.

Transport Layer: This is where TCP operates. It establishes a connection between two devices, manages data flow, and ensures data integrity.
Internet Layer: IP operates here, dealing with packet routing and addressing.

Link Layer: This layer involves protocols related to the physical transmission of data over a network medium (e.g., Ethernet).

Working Mechanism
The process begins when an application wants to send data. The data is passed to the transport layer, where TCP breaks it into manageable packets, each with a header containing sequencing information. These packets are handed over to the internet layer, where IP adds its own header, including source and destination IP addresses.

The packets are then transmitted over the network. Each router along the path uses the IP header to decide where to send the packet next, ensuring it moves closer to its destination. At the destination, TCP reassembles the packets into the original message and passes it to the appropriate application.

Significance and Applications
TCP/IP is essential for the Internet's functioning. It allows diverse devices and networks to interoperate seamlessly, enabling applications like web browsing, email, file transfer, and streaming. Its robustness and scalability have made it the standard networking protocol suite worldwide.

HyperText Transfer Protocol (HTTP)
HTTP is an application-level protocol used primarily for transferring hypertext documents on the World Wide Web. Developed by Tim Berners-Lee in the early 1990s, HTTP has become the foundation of data communication on the web.

Key Concepts
Request-Response Model: HTTP operates on a client-server model where the client (e.g., a web browser) sends a request to the server, which then processes the request and returns a response. This model underpins all web interactions.

Statelessness: HTTP is stateless, meaning each request from a client to a server is independent. The server does not retain any information about previous requests. This simplifies server design but requires mechanisms like cookies and sessions to maintain state across multiple requests.

HTTP Methods
HTTP defines several methods for client-server communication:
GET: Requests data from a specified resource.
POST: Submits data to be processed to a specified resource.
PUT: Updates a current resource with new data.
DELETE: Deletes a specified resource.
HEAD: Similar to GET, but retrieves only the headers, not the body.
Headers and Body

HTTP messages consist of:
Headers: Provide metadata about the request or response (e.g., content type, content length, and encoding).
Body: Contains the data being transmitted (optional, depending on the method).

HTTP Versions
HTTP/1.0: The initial version used basic request-response communication.
HTTP/1.1: Introduced persistent connections (allowing multiple requests and responses between a client and server), chunked transfer encoding, and additional cache control mechanisms.
HTTP/2: Improved performance by enabling multiplexing (sending multiple requests and responses simultaneously over a single connection), header compression, and more efficient use of network resources.
HTTP/3: Uses QUIC (Quick UDP Internet Connections) instead of TCP, providing faster connection establishment, improved security, and better performance over high-latency networks.

Security with HTTPS
HTTPS (HTTP Secure) is an extension of HTTP that uses SSL/TLS to encrypt data transmitted between a client and a server. This ensures data integrity, confidentiality, and authentication, protecting against eavesdropping and man-in-the-middle attacks.

Secure Sockets Layer/Transport Layer Security (SSL/TLS)
SSL and its successor TLS are cryptographic protocols designed to secure communication over a computer network. SSL was developed by Netscape in the mid-1990s to protect Internet communications. TLS, developed as an enhancement to SSL, provides improved security features and is the modern standard for secure communication.

Key Features
Encryption: Encrypts data to ensure confidentiality. Even if intercepted, the data cannot be read without the appropriate decryption key.
Authentication: Uses digital certificates to verify the identities of the communicating parties, ensuring that data is sent to and received from the correct source.
Integrity: Ensures that data is not tampered with during transmission. Any alterations can be detected.

How SSL/TLS Works
The SSL/TLS handshake process establishes a secure connection between a client and server:

Client Hello: The client sends a hello message to the server, specifying supported encryption algorithms and other settings.
Server Hello: The server responds with its chosen encryption algorithms and provides its digital certificate.

Certificate Verification: The client verifies the server's certificate against a list of trusted Certificate Authorities (CAs). If valid, the process continues.

Key Exchange: The client and server exchange cryptographic keys to establish a shared secret used for symmetric encryption.
Secure Connection: With the handshake complete, the secure connection is established, and encrypted data transmission begins.

TLS Versions
SSL 2.0 and SSL 3.0: Early versions with significant security flaws, now deprecated.
TLS 1.0: Improved upon SSL 3.0, addressing several vulnerabilities.
TLS 1.1 and TLS 1.2: Introduced stronger encryption algorithms and additional security features.
TLS 1.3: Simplified the handshake process and enhanced security and performance. It eliminates outdated cryptographic algorithms and reduces latency by combining steps in the handshake process.

Applications and Importance
SSL/TLS is crucial for securing various types of internet communication:
Web Browsing: HTTPS secures data between web browsers and servers, protecting against eavesdropping and tampering.
Email: Protocols like SMTPS, IMAPS, and POP3S use TLS to secure email communication.
VPNs: TLS can be used to secure virtual private networks (VPNs), ensuring secure remote access to network resources.
VoIP: Secures voice over IP (VoIP) communications, protecting against interception and eavesdropping.
Integration of TCP/IP, HTTP, and SSL/TLS
The interaction between TCP/IP, HTTP, and SSL/TLS protocols creates a comprehensive framework for internet communication:
TCP/IP: Provides the fundamental transport and routing mechanisms, ensuring data can be transmitted across diverse networks reliably.
HTTP: Sits atop TCP/IP, enabling the transfer of hypertext documents and facilitating the functioning of the World Wide Web.
SSL/TLS: Integrates with HTTP to form HTTPS, ensuring that data transmitted over the web is encrypted and secure.
In practical terms, when a user accesses a secure website:

TCP/IP Connection: The user's device establishes a TCP connection with the web server using IP addressing and routing.
SSL/TLS Handshake: The device and server perform an SSL/TLS handshake to create a secure connection.
HTTPS Request: The device sends an HTTPS request over the secure connection. HTTP operates as usual, but the underlying transport layer is encrypted by SSL/TLS.
Data Transmission: The server processes the request and sends an encrypted response back to the user's device. The device decrypts the response and displays the web page.
The protocols TCP/IP, HTTP, and SSL/TLS are foundational to the functioning and security of modern Internet communication. TCP/IP ensures reliable and efficient data transport across diverse networks. HTTP facilitates the transfer of hypertext documents, enabling the World Wide Web. SSL/TLS secures these communications, ensuring data integrity, confidentiality, and authentication. Together, these protocols create a robust and secure framework that supports the vast array of online activities we rely on daily, from browsing and emailing to streaming and online transactions. As technology evolves, these protocols continue to adapt, ensuring the internet remains a secure and efficient platform for global communication and information exchange.

Self-Assessment Exercise(s)

(1) Which protocol is primarily used for secure communication over the World Wide Web?
A) FTP
B) HTTP
C) HTTPS
D) SMTP
Answer: C) HTTPS

(2) What does the acronym FTP stand for, and what is its primary purpose?
A) File Transfer Protocol; used for transferring files between computers
B) Fast Transmission Protocol; used for quick data transmission
C) File Tracking Protocol; used for tracking file usage
D) File Transfer Pathway; used for creating pathways for file transfers
Answer: A) File Transfer Protocol; used for transferring files between computers

(3) Which protocol is used for sending emails from a client to a server?
A) IMAP

B) POP3
C) SMTP
D) HTTP
Answer: C) SMTP

(4) What is the primary function of the TCP/IP protocol suite?
A) To provide a set of rules for file transfer over the internet
B) To establish a connection-oriented communication service
C) To manage domain names on the internet
D) To enable internet routing and data packet delivery
Answer: D) To enable internet routing and data packet delivery

(5) Which of the following services is primarily used for real-time communication over the internet?
A) Email
B) File Transfer Protocol (FTP)
C) Voice over Internet Protocol (VoIP)
D) HyperText Transfer Protocol (HTTP)
Answer: C) Voice over Internet Protocol (VoIP)

Conclusion

Internet services, communication, and protocols are integral components that collectively ensure the functionality and efficiency of the Internet. The vast array of internet services enables users to perform a multitude of tasks, from browsing the web and sending emails to streaming media and engaging on social platforms. These services rely on effective communication facilitated by a suite of protocols that govern data exchange, ensuring seamless interaction between disparate systems and devices. Protocols like TCP/IP, HTTP, SMTP, and FTP provide the necessary framework for reliable and accurate data transmission, underpinning the robustness and interoperability of the Internet. Together, these elements create a cohesive and dynamic digital ecosystem that supports global connectivity and information exchange.

4.0 Summary

Internet services, communication, and protocols form the backbone of the modern Internet, enabling a wide range of activities and applications. Internet services refer to various online services such as web browsing, email, file transfer, streaming, and social media, provided by different platforms and servers over the Internet. These services rely on a network of interconnected devices, including computers, servers, and other smart devices, to exchange information and provide users with access to

digital resources. The interaction between users and these services is facilitated by web browsers, email clients, media players, and other applications designed to connect to the internet and handle specific types of data.

Communication on the Internet is governed by a set of rules and standards known as protocols. These protocols ensure that data is transmitted accurately and efficiently between devices. Key protocols include the Transmission Control Protocol (TCP) and the Internet Protocol (IP), which work together to route data packets across networks and ensure they reach their intended destinations. Other important protocols include Hypertext Transfer Protocol (HTTP) for web traffic, Simple Mail Transfer Protocol (SMTP) for email, and File Transfer Protocol (FTP) for file transfers. These protocols define how data is formatted, transmitted, and received, allowing different systems to communicate seamlessly, regardless of their underlying hardware or software differences. By adhering to these standardized protocols, the internet maintains its robustness, reliability, and interoperability.

5.0 References/Further Reading

Comer, D. E. (2018). The Internet book: everything you need to know about computer networking and how the Internet works. Chapman and Hall/CRC.

Hersent, O., Boswarthick, D., & Elloumi, O. (2011). The internet of things: Key applications and protocols. John Wiley & Sons.

Ciubotaru, B., Muntean, G. M., Ciubotaru, B., & Muntean, G. M. (2013). Network communications protocols and services. Advanced Network Programming–Principles and Techniques: Network Application Programming with Java, 29-52.

Ciubotaru, B., Muntean, G. M., Ciubotaru, B., & Muntean, G. M. (2013). Network communications protocols and services. Advanced Network Programming–Principles and Techniques: Network Application Programming with Java, 29-52.

Unit 4 Web Application Development

Contents

1.0 Introduction

A web application is a software program that runs directly in a web browser. Unlike traditional desktop applications, it doesn't require downloads or installations. Many types of web applications are built using client-side scripts (like HTML, JavaScript, or CSS) and server-side scripts (like PHP or ASP). They interact with users by sending requests to the server, which processes the data and returns the results. This setup allows web applications to perform various functions, from displaying content to managing user data. They're accessible from any device with an internet connection, making them versatile for many uses, from personal blogs to complex e-commerce sites. Web app development services typically follow a meticulous process in creating a custom web application, involving careful planning, design, coding, testing, and maintenance.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

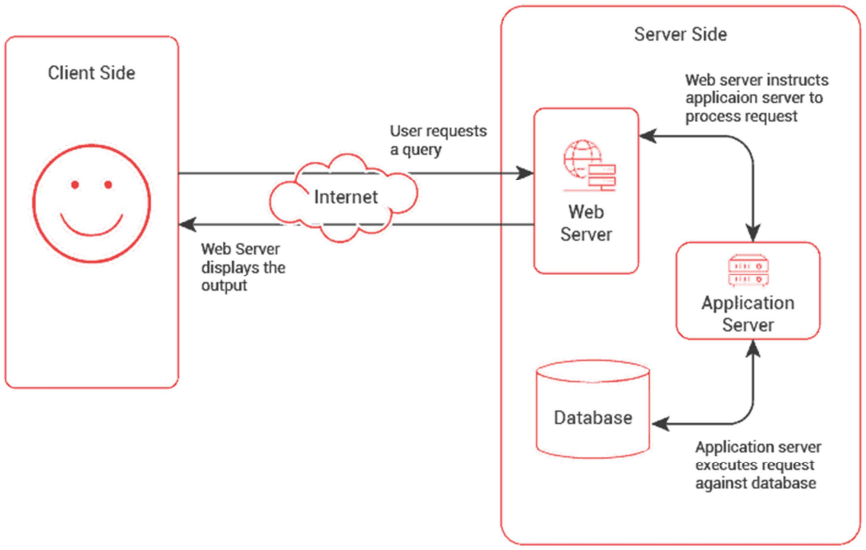
- Understand Internet.
- Identify various services the Internet provides for the users.
- Explain the Internet communication channels
- Explain some common Internet Protocols

3.0 Main Content

3.1 Overview

A decade ago, web pages were mostly static, with a few images and videos scattered around. In 2005, thanks to Ajax, it made it possible to create better, faster, and more interactive web applications. A web application, also known as a web app, is nothing but a computer program that uses a web browser to perform a specific function. A web application is a client-server program that comprises a client-side and a server-side. The user enters data through the client side (front-end), while the server side (back-end) of the application stores and processes the information. For example, shopping carts, content management systems, and online forms are typical web applications. Both enterprises and individuals develop web applications to fulfill different purposes. Web apps help integrate the tailored experience of native apps with easy access on a website browser from any device. For example, LinkedIn, Basecamp, MailChimp, and even Facebook, have web apps that provide immersive and tailored experiences like the native apps directly from the browser. Hence, web application development is gaining a lot of popularity in almost all industries, including banking, eCommerce, education, healthcare, and more.

How does a Web Application Work?
Web applications are accessed over a network and need not be downloaded. Instead, users can access web applications through browsers like Google Chrome, Mozilla Firefox, Opera, or Safari. A web application is built around three components- a Web Server, an Application Server, and a Database. The web server manages requests from the client, the application server processes requests, and the database stores the information. A typical web application workflow looks like this: 1 User triggers a request to the web server, either through a web browser or



3.2 Types of Web Applications

Now that we've covered the various aspects of web applications let's deep dive into the different categories of web applications. Web apps are classified based on their functionalities, tools, and technologies.

1. Static Web Applications

   Static web applications are simple web-based applications that deliver pre-rendered content to users without server-side processing or dynamic content generation. Each page within a static web application is fixed and does not change in response to user interactions or data inputs. These applications consist of HTML, CSS, and JavaScript files that are directly served to the client's browser. As a result, static web applications are fast, secure, and easy to deploy, but they lack the interactivity and real-time data manipulation capabilities of dynamic web applications. They are best suited for use cases like personal blogs, portfolios, and informational websites where content changes infrequently. These are the simplest types of web applications built using HTML and CSS, suitable for creating portfolios or digital brochures. As the name suggests, the content only changes if manually updated by the developer. Static web applications are straightforward to create and host, as they don't require extensive server-side processing. This option is a cost-effective solution for individuals or small businesses needing a simple online presence. However, their simplicity also means limited functionality.

2. Dynamic Web Applications

   Dynamic web applications are sophisticated web-based applications that generate content dynamically based on user interactions, inputs, or real-time data. Unlike static web applications, they involve server-side processing, where scripts (such as PHP, Python, Ruby, or Node.js) run on the server to construct web pages on-the-fly before sending them to the client's browser. This allows for interactive features, personalized content, and the ability to connect to databases to retrieve and update information. Dynamic web applications are used for a wide range of purposes, including social media platforms, e-commerce sites, and online forums, where user engagement and up-to-date information are crucial. They offer a highly interactive user experience but require more complex development and server resources compared to static web applications. In contrast, dynamic web applications are more complex and interactive. They use client-side and server-side scripts (like JavaScript, PHP, ASP, or JSP) to generate content in real-time. These web application categories are connected to a database, allowing them to provide personalized experiences based on user interactions and preferences. They're ideal for businesses, especially if your top priorities are user engagement and content variability. Due to their complexity, dynamic web applications are more challenging to develop and maintain. They require a more robust hosting environment and higher web development costs.

3. Single-Page Applications (SPAs)

   SPAs are advanced web applications that provide a seamless, dynamic user experience by loading a single HTML page and dynamically updating content as the user interacts with the app. Unlike traditional multi-page applications, SPAs avoid full page reloads by using JavaScript frameworks such as Angular, React, or Vue.js to manage the user interface and handle client-side routing. This approach enhances performance and responsiveness, creating an experience similar to that of a desktop application. SPAs rely on AJAX calls to communicate with the server in the background, fetching only the necessary data and updating the page accordingly. This results in faster transitions and a more fluid user experience, making SPAs ideal for applications requiring a high degree of interactivity, such as email clients, social media platforms, and complex dashboards. SPAs load a single HTML page and dynamically update content as users interact with the app. These categories of web applications are ideal for platforms where user experience and speed are critical, such as:

i. Social media platforms
ii. Email clients
iii. Cloud-based software

The benefit is that this web app type avoids reloading the entire page with each user action, leading to a smoother and faster user experience. However, they also come with challenges, particularly in SEO optimization and initial load times, as the entire application must be loaded simultaneously. SPAs are built using JavaScript frameworks like Angular, React, or Vue.js, which handle the dynamic loading of content and user interface elements.

4. Multi-Page Web Applications (MPAs)
MPAs are traditional web applications where each interaction or request from the user results in the loading of a new page from the server. This architecture involves multiple HTML pages, each corresponding to different views or sections of the application. When a user navigates or performs an action, the browser makes a request to the server, which processes it, retrieves or updates data as needed, and then sends back a fully rendered HTML page. MPAs are well-suited for complex applications with extensive content and varied navigation paths, such as e-commerce websites, blogs, and corporate portals. While MPAs can be less fluid than Single-Page Applications (SPAs) due to frequent page reloads, they offer advantages in terms of SEO and initial load times, as well as easier implementation of analytics and tracking. Unlike SPAs, MPAs reload the entire page from the server when the user interacts with the application. These traditional web application categories are more suitable for websites with a large amount of content and diverse functionalities, such as:
eCommerce sites
Online catalogs
Educational platforms

MPAs can handle complex structures and vast databases more efficiently than SPAs. They're also better optimized for search engines, as each page can be indexed separately. However, MPAs often have slower page transitions and can be more resource-intensive, as each new page needs a server request and page reload. Developing MPAs typically involves a more extensive back-end process to manage multiple pages and their interactions with the server. When considering developing these web application types, you can partner with a skilled website

development company to ensure a seamless and effective online presence.

5. Progressive Web Applications (PWAs)

PWAs are a type of web application that leverages modern web capabilities to deliver an app-like experience to users. They combine the best features of web and mobile applications, offering fast load times, offline functionality, and push notifications. PWAs are built using standard web technologies such as HTML, CSS, and JavaScript, and are designed to be responsive, reliable, and installable. They can be added to a user's home screen without needing to go through an app store, and they use service workers to manage caching and offline access, ensuring that the app remains functional even without a network connection. By providing a seamless, engaging user experience across various devices, PWAs bridge the gap between traditional web pages and native mobile apps. PWAs represent a hybrid of regular web pages (or websites) and a mobile application. They're installable on a device's home screen without downloading from an app store. A key feature of PWAs is the use of service workers or scripts running in the background, enabling:

Offline functionality

Push notifications

Background data syncing

PWAs are also responsive and linkable, which can be shared via a URL. They offer a high level of performance, engaging users with smooth animations and no janky scrolling. Thus, these categories of web applications are incredibly efficient, especially for users with limited internet connectivity. Developers use standard web technologies to build them, including HTML, CSS, and JavaScript.

6. Content Management Systems (CMS)

CMS are software platforms designed to facilitate the creation, management, and modification of digital content without requiring extensive technical knowledge. They provide an intuitive interface for users to easily add, edit, and publish content, typically through a web-based editor. CMS platforms, such as WordPress, Joomla, and Drupal, often include tools for organizing content, managing media files, and handling user permissions. They support extensibility through plugins and themes, allowing users to customize the functionality and appearance of their websites. By abstracting much of the underlying code and technical complexities, CMS makes it accessible for non-developers to maintain dynamic and feature-rich websites, making them ideal for blogs, corporate websites, e-commerce sites, and online publications. A CMS manages the

creation and modification of digital content, supporting multiple users in a collaborative environment. CMS features vary widely, including but not limited to:

- Web-based publishing
- Format management
- History editing
- Version control
- Indexing
- Search

They're suitable for blogging, e-commerce, and news websites, where you need to frequently update the content without extensive technical know-how. CMS platforms like WordPress, Joomla, and Drupal are popular choices, offering templates and plugins for customization without needing to write code from scratch. What's more, CMSs provide a user-friendly interface, allowing for easy updates and managing content.

7. eCommerce Web Applications

eCommerce web applications are online platforms designed for buying and selling products or services over the Internet. They provide a comprehensive suite of features to facilitate transactions, including product catalogs, shopping carts, payment gateways, and order management systems. These applications often incorporate user account management, allowing customers to track orders, save payment information, and manage their preferences. eCommerce web applications are built to handle various aspects of online retail, such as inventory management, customer service, marketing tools, and analytics to track sales and user behavior. By offering a seamless shopping experience across multiple devices, these applications enable businesses to reach a broader audience, enhance customer engagement, and drive sales growth. Popular examples include platforms like Shopify, Magento, and WooCommerce. These web application types facilitate online buying and selling. They're complex systems that integrate various functionalities, including:

- Product displays or catalogs
- Product search and filtering
- Shopping carts
- Payment processing
- Customer account and order management
- Customer service tools

E-commerce web applications must provide a seamless, user-friendly experience to encourage sales and repeat business. These applications must be scalable to handle varying traffic and sales volume levels. Also, security is paramount to protect sensitive customer data, including payment information. Platforms

like Shopify, Magento, and WooCommerce are famous examples, offering customizable templates and various plugins to enhance functionality. E-commerce web applications have revolutionized the retail industry, allowing businesses to reach a wider audience and operate 24/7.

8. JavaScript-Powered Web Applications

JavaScript-powered web applications utilize JavaScript extensively to create interactive, dynamic, and responsive user experiences directly within the browser. These applications leverage JavaScript frameworks and libraries such as React, Angular, and Vue.js to manage the user interface, handle client-side logic, and facilitate asynchronous communication with servers through APIs. This approach enables real-time updates and smooth transitions without the need to reload the entire page, enhancing the overall performance and user experience. JavaScript-powered applications can range from single-page applications (SPAs) to more complex multi-page applications that require dynamic content rendering and user interactivity. By using JavaScript, developers can create rich web applications that are capable of mimicking the look and feel of native desktop or mobile apps while being accessible through web browsers. JavaScript is a versatile programming language to create dynamic and interactive user experiences. JavaScript can be used both on the client side (in the browser) and the server side (with technologies like Node.js), making it a powerful tool for full-stack development. JavaScript-powered web applications are known for their speed and efficiency; they can update content without reloading the entire page. This functionality makes them ideal for applications that require real-time data updates, such as:

Social media platforms

Online games

Collaboration tools

React, Angular, and Vue.js are examples of JavaScript frameworks and libraries. They provide pre-written JavaScript code to handle everyday tasks.

9. Rich Internet Web Applications (RIAs)

RIAs) are web-based applications that provide user experience similar to that of traditional desktop applications, featuring rich interactivity, responsive interfaces, and high levels of user engagement. RIAs leverage advanced web technologies such as HTML5, CSS3, JavaScript, and frameworks like Adobe Flash (historically), Silverlight, or modern JavaScript libraries like React and Angular to deliver sophisticated functionalities directly in the web browser. These applications support multimedia, real-

time data processing, and dynamic content updates without requiring full page reloads, ensuring smooth and seamless interactions. RIAs are commonly used in areas like online gaming, interactive data visualization, multimedia editing, and complex business applications, offering users a robust and immersive experience that goes beyond the capabilities of traditional web applications. RIAs are advanced web application types that deliver a user experience like desktop applications. They use client-side frameworks to provide interactive features and a richer user interface, such as:
Adobe Flash
JavaFX
Microsoft Silverlight
RIAs run inside web browsers but behave like desktop applications, offering responsive, engaging user experience with better data visualization and real-time interaction capabilities. RIAs can process data and perform tasks without constantly communicating with the server, reducing load times and improving performance. However, they require plugins or specific frameworks, which might limit accessibility and compatibility across different devices and browsers.

10. Portal Web Applications
Portal web applications serve as centralized access points that aggregate a wide range of information, services, and resources, often catering to specific user groups or organizational needs. These applications provide personalized user interfaces where users can log in to access customized content, tools, and applications based on their roles and preferences. Portal web applications typically integrate various functionalities such as email, forums, search engines, dashboards, and news feeds, facilitating seamless navigation and interaction within a single cohesive platform. They are widely used in corporate environments, educational institutions, and governmental organizations to streamline workflows, enhance communication, and provide a unified user experience. By consolidating disparate resources into a single entry point, portal web applications improve accessibility and efficiency for users seeking comprehensive and tailored information. Portal web applications are gateways to various information, services, and other applications. Enterprises often use them for internal purposes or to provide customer-facing services, for example:

These web application types typically require user authentication, offering personalized content and a centralized access point to various resources. Portal web applications are designed to

aggregate content from different sources, providing a consistent and integrated user experience. They can handle various functionalities, including:
Search engines
Email systems
Forums
Newsfeeds

11. Animated Web Applications
Animated web applications incorporate dynamic visual elements, transitions, and effects to enhance user engagement and interaction. These applications leverage technologies such as CSS animations, JavaScript libraries like GSAP (GreenSock Animation Platform), or frameworks like WebGL to create fluid animations and interactive experiences directly within the browser. Animated web applications utilize motion to convey information, guide users through workflows, and provide feedback on user actions, resulting in a more intuitive and enjoyable user experience. Whether through subtle micro-interactions or complex animated interfaces, these applications captivate users' attention, communicate messages effectively, and differentiate themselves in a crowded digital landscape. These applications focus on delivering rich visual content and interactive elements using animations. They're particularly popular in fields that need high levels of user engagement, such as:
- Online advertising
- Gaming
- Educational platforms

Animated web apps are built using technologies like CSS3, HTML5, and WebGL, so developers can create complex yet engaging user interfaces. They provide dynamic and visually appealing experiences, capturing users' attention and improving interaction. However, developing these applications can be time-consuming and require advanced design as well as programming skills. Furthermore, you need to balance the animations to enhance rather than hinder the user experience, especially considering performance and accessibility on various devices.

3.3 Web application development process

The number of steps in the web application development process can vary between 5 and 9. But on average, 7 crucial steps are involved in the web app development model. The same are as follows:

- Requirements review & proposal
- Planning & blueprints
- Web application design
- Copywriting & labeling
- Web application programming
- Testing & Launch
- Application maintenance

Stage1: Requirements review & proposal
Entrepreneurs and businesses mostly start with a set of ideas when they think of launching a web application. These ideas slowly evolve into a detailed document in which application goals, features, technology, budget, vision, and plans are listed out.

By going through this document, the development team gets a clear understanding of your app objectives, key goals, target audience, focus industry, milestones, and other critical elements. This document is followed up by discussions and questionnaires that help web developers get further clarity into project goals.

Once the application development team has 100% clarity on everything project-related, the proposal is prepared to document everything that will be delivered.

Stage 2: Planning & blueprints
In Stage 1, both teams reached an understanding about the envisioned web application. Now, it's time to create the roadmap that will be followed to build it. With the help of insights gathered in the previous stage of the web app development model, developers create a blueprint including flowcharts and sketches that helps determine the overall structure of the web application.

Flowcharts - also known as Sitemaps - show the relationship between different web pages and help understand how the inner structure of your website will look & work. Wireframes are often used for a visual representation of the UI. Top web app development teams keep clients in loop during this stage to make sure that the core of the application comes out correctly.

Depending on the complexity of the web application, the time spent on Stage 2 will vary. If creating a Minimum Viable Product, be prepared to dedicate around 2 weeks on this.

Stage 3: Web application design
Web application users don't know what happens behind the frontend of the app and how things work. All they interact with is the design part of the application. Stage 3 is all about perfecting the design & interactive elements of the web application. Designers work with color schemes, graphics, icons, templates, user experience, style guides, transitions, buttons, and much more to finalize the design aspect of the web application.

After finalizing the initial mockups, they are shared with clients for review & feedback. The design iterations and mockup changes go on until the client gives a thumbs-up to everything. While the design team is busy with the mockups and refinement, the development team is mostly engaged with the programming part. So, Stage 3 and Stage 4 usually move ahead alongside each other.

Stage 4 - Web application programming
If Stage 3 is crafting the aesthetics of a car, Stage 4 is about putting together the engine. App programming makes the envisioned features function and builds the value component for the customers. In this stage, frameworks are developed, APIs deployed, app features built, security layers added, payment gateways integrated, and lots of other capabilities crafted.

While coding complex web applications is a time-consuming process, a lot depends on the technologies opted. Some technology stacks benefit from libraries that have capabilities that can be tweaked and integrated with ease. A lot also depends on the experience and expertise of the web programmers working on your web application. Stage 4 forms the biggest chunk of the web application model!

Stage 5 - Copywriting & labeling
Copy and labeling are less than 5% of the application development work but without it, it is hard to make sense of everything you have built. User experience and user interface greatly depend on the talent of the Information Architect and Copywriter engaged for the project. Usability and simplicity must be the epicenter of this step in the web app development process.

Stage 5 is about finalizing the headlines, captions, labeling, copy, and everything else that's in the text form. The collaboration of the designer,

developer, copywriter, and IA is critical to executing all the copy in the right place!

Stage 6: Testing & Launch

Testing the application after everything seems good to go is the most important aspect of the web app development model. That's because there are hundreds of things that can go wrong even after you think every inch of the application has been executed correctly.

Start by checking:

- Core features
- Forms
- Links
- Buttons
- Upload functionality (if any)
- Copy
- Transitions
- Performance

Even after double-testing everything, it is a good idea to launch your web application initially in the beta version. If the stakes are high and resources are limited, the web application can be unfolded in phases to different audience groups.

Stage 7: Application maintenance

Be it a simple business website or a complex web application, every digital product need routine checkups and enhancements. With the passage of time, you will want to undertake product pivots, integrate new features, and launch the App Version 2. This is why your app development agreement should talk about application maintenance, after-delivery support, and future upgrades.

3.4 Web Application Development Validation and Deployment

Now that we have discussed web app development and how it works, let us take a look into other aspects, such as validation and deployment. These two are some of the most critical aspects of developing a website or web portal. Let us look into the same.

Once your web application has been created and is prepared for release, testing is essential to ensure it runs properly before deployment. Fixing bugs is insufficient on its own. As a result, testing is essential to the creation of online applications.

A standard web application undergoes the following testing:

Usability testing

Usability testing for web applications involves evaluating a website's design, functionality, and overall user experience by observing real

users as they interact with the site. The goal is to identify any usability issues, such as confusing navigation, unclear instructions, or cumbersome workflows, that may hinder users from effectively accomplishing their tasks. By conducting usability testing, developers and designers gain insights into user behavior, preferences, and pain points, enabling them to make data-driven improvements to enhance the website's accessibility, efficiency, and satisfaction. This process typically includes various techniques like task-based testing, user interviews, and heuristic evaluations, ensuring the final product meets user needs and expectations.

Performance testing

Performance testing for web applications involves evaluating the application's speed, responsiveness, and stability under various conditions to ensure it can handle expected and peak user loads effectively. This testing aims to identify performance bottlenecks, such as slow page load times, server response delays, and resource utilization issues, which can negatively impact the user experience. Techniques such as load testing, stress testing, and scalability testing are employed to simulate different user scenarios and traffic levels. By conducting performance testing, developers can optimize the web application's infrastructure, improve its efficiency, and ensure it remains reliable and fast, even during high-traffic periods, thus providing a smooth and satisfactory user experience.

Web App Security testing

Web app security testing involves evaluating a web application's defenses against potential threats and vulnerabilities to ensure it can protect sensitive data and maintain integrity, confidentiality, and availability. This testing includes identifying and mitigating security flaws such as cross-site scripting (XSS), SQL injection, cross-site request forgery (CSRF), and other common exploits that attackers may use. Techniques like penetration testing, vulnerability scanning, and code reviews are employed to uncover and address security weaknesses. By conducting comprehensive security testing, developers can fortify the application against unauthorized access, data breaches, and other cyber threats, ensuring a secure and trustworthy environment for users.

Quality assurance

Quality assurance (QA) testing for web applications is a comprehensive process that ensures the application meets specified requirements and standards, providing a high-quality user experience. This involves systematic testing of all aspects of the application, including functionality, usability, performance, security, and compatibility across different devices and browsers. QA testing methods include manual testing, automated testing, regression testing, and user acceptance

testing (UAT), aimed at identifying and fixing defects early in the development cycle. By rigorously validating the application's features and behavior, QA testing helps deliver a reliable, efficient, and user-friendly web application, ultimately boosting user satisfaction and reducing the risk of post-deployment issues.
Testing & bug fixing

Testing and bug fixing in web applications is a critical phase in the development lifecycle aimed at ensuring the application operates correctly and efficiently. This process involves systematically identifying, documenting, and resolving defects or issues found during various testing stages, such as functional, usability, performance, and security testing. Testing helps uncover problems that may impact the user experience or application performance, while bug fixing involves addressing these issues through code corrections, adjustments, and optimizations. Effective testing and prompt bug fixing are essential for maintaining the application's stability, reliability, and quality, ensuring that users encounter minimal errors and enjoy a seamless interaction with the web application.

Browser compatibility testing
Browser compatibility testing for web applications involves evaluating the application's performance and appearance across different web browsers and their various versions to ensure consistent functionality and user experience. This testing checks that web elements such as layouts, fonts, images, and scripts render correctly and work seamlessly, regardless of the browser being used. It addresses discrepancies caused by differences in browser engines, HTML/CSS interpretations, and JavaScript execution. By conducting thorough browser compatibility testing, developers can identify and fix issues specific to certain browsers, ensuring that the web application is accessible, visually consistent, and fully operational for all users, regardless of their preferred browsing platform.

Responsive testing
Responsive testing for web applications involves verifying that the application's design and functionality adapt seamlessly across various devices and screen sizes, including desktops, tablets, and smartphones. This testing ensures that the user interface elements, such as navigation menus, images, text, and interactive components, are displayed correctly and are usable on different screen resolutions and orientations. By simulating different device environments and screen sizes, responsive testing identifies layout issues, performance inconsistencies, and usability problems that could affect the user experience. Ensuring responsiveness is crucial for delivering a flexible, user-friendly web

application that provides an optimal experience regardless of the device used.

3.5 Benefits of Web Applications

In today's fast-paced digital landscape, businesses are continually seeking innovative ways to stay competitive and connect with their target audiences. **Web application development** has emerged as a powerful tool that enables companies to enhance their online presence and streamline various business processes. In this article, we will explore the top seven benefits of web application development for businesses and delve deeper into each of them.

Global Reach and Accessibility
One of the primary advantages of web applications is their ability to provide global reach and accessibility. Unlike traditional desktop applications that are confined to a specific device or location, web applications can be accessed from any device with an internet connection and a compatible web browser. This means that businesses can reach a global audience without the need for users to install or update software.

Benefit: Web applications provide a consistent and easily accessible platform for engagement, ensuring that your business can connect with customers across the street or on the other side of the world.

Cost-Effective Solution
Web application development offers a cost-effective solution for businesses of all sizes. Unlike native mobile applications that require separate development efforts for various platforms (iOS, Android, etc.), web applications are platform-agnostic. This means that a single web application can run on multiple devices and operating systems without the need for extensive customization.

Benefit: The cost-effectiveness of web applications reduces development expenses, allowing businesses to allocate resources more efficiently and maximize their return on investment (ROI).

Scalability and Flexibility
As businesses grow and evolve, their software needs often change. Web applications are highly scalable and flexible, allowing them to adapt to changing requirements and increasing user loads. Whether you need to add new features, accommodate a larger user base, or integrate with third-party services, web applications can be easily modified and expanded.

Benefit: Scalability ensures that your software investment can grow alongside your business, saving you the hassle and expense of reinventing your entire application as your needs change.

Cross-Platform Compatibility
Web applications are designed to be cross-platform compatible, meaning they can run seamlessly on various devices and operating systems. Whether your users prefer Windows, macOS, iOS, Android, or other platforms, they can access your web application with ease. This versatility eliminates the need to develop and maintain separate applications for different platforms.

Benefit: Cross-platform compatibility simplifies the development process, reduces costs, and ensures a consistent user experience across devices, fostering user satisfaction and loyalty.

Real-Time Updates and Maintenance
Web applications are hosted on web servers, allowing for real-time updates and maintenance. This means that businesses can quickly address bugs, security vulnerabilities, or performance issues without inconveniencing users. Users don't need to manually download and install updates, as web applications are automatically updated when they access the latest version of the application.

Benefit: Real-time updates and maintenance ensure that your web application remains secure and up-to-date, providing users with a reliable and hassle-free experience.

Enhanced Security
Security is a top priority for businesses, especially when handling sensitive data or transactions. **Web application development offers robust security features** to protect both your business and your users. With the right security measures in place, web applications can safeguard against common threats such as data breaches, unauthorized access, and malware.

Benefit: Enhanced security measures instill trust in users and protect your business from potential threats, ensuring the confidentiality and integrity of sensitive information.

Improved Analytics and Insights
Web applications provide valuable data and insights that can inform business decisions and strategies. Through integrated analytics tools, businesses can track user behavior, monitor engagement, and gather actionable data. This information helps in understanding user

preferences, identifying areas for improvement, and optimizing the user experience.

Benefit: Improved analytics and insights enable data-driven decision-making, allowing businesses to adapt and refine their strategies to meet user needs and drive growth.

Conclusion

In conclusion, web application development, whether done in-house or by partnering with a reputable **web application development company**, offers a wide range of benefits for businesses looking to thrive in the digital era. The global reach and accessibility, cost-effectiveness, scalability and flexibility, cross-platform compatibility, real-time updates and maintenance, enhanced security, and improved analytics and insights empower businesses to connect with their audience, streamline operations, and stay ahead of the competition.

As technology continues to advance, embracing web application development is a strategic choice that can drive innovation, efficiency, and success in today's competitive business landscape. By harnessing the power of web applications, businesses can position themselves for growth, engage with their customers effectively, and adapt to the ever-changing digital environment. Whether you're a startup or an established corporation, web application development, with the right development partner, can be a game-changer in achieving your business goals.

Self-Assessment Exercise(s)

(1) Which of the following is a common front-end technology used in web development?
A) Python
B) Node.js
C) React
D) Django
Answer: C) React

(2) Which HTTP method is typically used to retrieve data from a web server?
A) POST
B) PUT
C) DELETE
D) GET
Answer: D) GET

(3) What does REST stand for in the context of web services?
A) Reliable Event Service Transmission

B) Representational State Transfer
C) Rapid Exchange Service Technology
D) Remote Execution and Synchronization Transfer
Answer: B) Representational State Transfer

(4) Which of the following is a popular database management system used in web applications?
A) MongoDB
B) MATLAB
C) Apache Hadoop
D) TensorFlow
Answer: A) MongoDB

(5) In the context of web applications, what is the purpose of an API?
A) To analyze application performance
B) To provide a user interface for the application
C) To facilitate communication between different software components
D) To secure the application against cyber attacks
Answer: C) To facilitate communication between different software components

Conclusion

Web application development refers to the process of using client-side and server-side programming to develop an application that is accessible over the web browser. The web application development process begins by; first, the developer trying to find a solution to a specific problem, then designing the web app by choosing the appropriate development framework. Next, the developer tests the solution and finally deploys the web app.

4.0 Summary

A web application is an application program that is stored on a remote server and delivered over the internet through a browser interface. Web services are web apps by definition and many, although not all, websites contain web apps. Developers design web applications for a wide variety of uses and users, from an organization to an individual for numerous reasons. Commonly used web applications can include webmail, online calculators, or e-commerce shops. While users can only access some web apps by a specific browser, most are available no matter the browser.

5.0 References/Further Reading

Sajja, P. S., & Akerkar, R. (2012). Intelligent technologies for Web applications. CRC Press.

Su, J. M. (2023). WebHOLE: Developing a web-based hands-on learning environment to assist beginners in learning web application security. Education and Information Technologies, 1-32.

Westfall, J., Augusto, R., & Allen, G. (2012). Beginning Android Web Apps Development: Develop for Android Using HTML5, CSS3, and JavaScript. Apress.

MODULE 2 HTML FUNDAMENTALS

MODULE INTRODUCTION

Hypertext Markup Language (HTML) is the backbone of the World Wide Web, serving as the standard language used to create and design web pages. Understanding HTML is crucial for anyone looking to build websites or delve into web development. It provides the structure of a webpage by using elements and tags to define content such as headings, paragraphs, links, images, and more. This module, titled "HTML Fundamentals," aims to introduce the core concepts and syntax of HTML, equipping learners with the foundational knowledge needed to create well-structured and functional web pages. In this module, students will explore the essential elements of HTML, starting with the basic document structure and gradually progressing to more complex components like forms, tables, and multimedia integration. Through a series of hands-on exercises and practical examples, learners will gain experience in coding and debugging HTML documents. By the end of the module, participants will be proficient in using HTML to build static web pages, understand best practices for web development, and be prepared for more advanced topics in web design and development.

Unit 1 Introduction to HTML

Contents

1.0 Introduction

Hypertext Markup Language (HTML) is the standard markup language used to create web pages. It's a combination of Hypertext, which defines the link between web pages, and Markup language, which is used to define the text document within tags to structure web pages. This language is used to annotate text so that machines can understand and manipulate it accordingly. HTML is human-readable and uses tags to define what manipulation has to be done on the text. This unit will help you understand the workings of HTML and explain it with examples.

2.0 Intended Learning Outcomes (ILOs)

Understanding Basic HTML Structure

- Explain the purpose of common HTML tags and attributes
- Identify and describe the usage of various HTML elements
- Develop a simple web page using basic HTML elements

3.0 Main Content

3.1 A Brief History of HTML

HTML is a markup language used by the browser to manipulate text, images, and other content, in order to display it in the required format. HTML was created by Tim Berners-Lee in 1991. The first-ever version of HTML was HTML 1.0, but the first standard version was HTML 2.0, published in 1995.

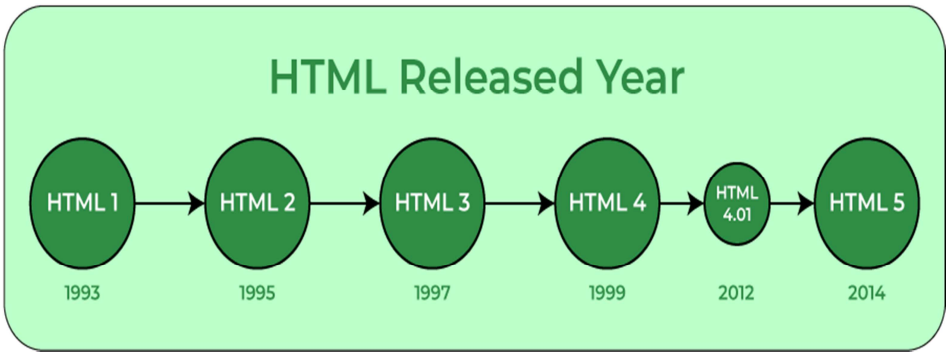
The first version of HTML was written by Tim Berners-Lee in 1993. Since then, there have been many different versions of HTML. The most widely used version throughout the 2000's was HTML 4.01, which became an official standard in December 1999.

Another version, XHTML, was a rewrite of HTML as an XML language. XML is a standard markup language that is used to create other markup languages. Hundreds of XML languages are in use today, including GML (Geography Markup Language), MathML, MusicML, and RSS (Really Simple Syndication). Since each of these languages was written in a common language (XML), their content can easily be shared across applications. This makes XML potentially very powerful, and it's no surprise that the W3C would create an XML version of

HTML (again, called XHTML). XHTML became an official standard in 2000, and was updated in 2002. XHTML is very similar to HTML, but has stricter rules. Strict rules are necessary for all XML languages, because without it, interoperability between applications would be impossible. You'll learn more about the differences between HTML and XHTML in Unit 2.

Most pages on the Web today were built using either HTML 4.01 or XHTML 1.0. However, in recent years, the W3C (in collaboration with another organization, the WHATWG), has been working on a brand new version of HTML, HTML5. Currently (2011), HTML5 is still a draft specification, and is not yet an official standard. However, it is already widely supported by browsers and other web-enabled devices, and is the way of the future. Therefore, HTML5 is the primary language taught in this course.

Examples of types of content that can be included on web pages
The following table shows a list of many of the types of content that can be added to web pages using different versions of HTML. In the early days of the Web, HTML (version 1.2) was very simple, but over time new versions were released that added more and more features. Still, if web designers wanted to add content or features that HTML didn't support, they would have to do so with non-standard proprietary technologies such as Adobe Flash. These technologies would require users to install browser plug-ins, and in some cases meant that certain users would be unable to access the content (for example, iPhones and iPads don't support Flash).



HTML5 has added support for many new features that will make it possible to do more with HTML, without relying on non-standard proprietary technologies.

Type of content	**HTML 1.2**	**HTML 4.01**	**HTML5**	**Purpose**
Heading	Yes	Yes	Yes	Organize page content by adding headings and subheadings to the

				top of each section of the page
Paragraph	Yes	Yes	Yes	Identify paragraphs of text
Address	Yes	Yes	Yes	Identify a block of text that contains contact information
Anchor	Yes	Yes	Yes	Link to other web content
List	Yes	Yes	Yes	Organize items into a list
Image	Yes	Yes	Yes	Embed a photograph or drawing into a web page
Table	No	Yes	Yes	Organize data into rows and columns
Style	No	Yes	Yes	Add CSS to control how objects on a web page are presented
Script	No	Yes	Yes	Add Javascript to make pages respond to user behaviors (more interactive)
Audio	No	No	Yes	Add audio to a web page with a single tag
Video	No	No	Yes	Add video to a web page with a single tag
Canvas	No	No	Yes	Add an invisible drawing pad to a web page, on which you can add drawings (animations, games, and other interactive features) using Javascript

3.2 The Crucial Role of HTML in Web Development

At its heart, HTML is a language made up of elements, that can be applied to pieces of text to give them different meanings in a document

(Is it a paragraph? Is it a bulleted list? Is it part of a table?), structure a document into logical sections (Does it have a header? Three columns of content? A navigation menu?), and embed content such as images and videos into a page. This module will introduce the first two of these and introduce fundamental concepts and syntax you need to know to understand HTML.

HTML plays an essential role in web development as it defines the structure and content of web pages. It serves as the backbone upon which websites are built. HTML accomplishes this by utilizing a system of tags and elements, each serving a unique purpose. Tags are enclosed within angle brackets, each comprising an opening and closing part. They function as building blocks that define the structure of your web page. Think of them as the bricks and mortar of web development. Understanding their roles is essential for web development.

It is the most basic language, and simple to learn and modify. It is a combination of both hypertext and markup language. It contains the elements that can change/develop a web page's look and the displayed contents. Or we can say that HTML creates or defines the structure of web pages. We can create websites using HTML which can be viewed on internet-connected devices like laptops, android mobile phones, etc. It was created by Tim Berners-Lee in 1991. The first version of HTML is HTML 2.0 which was published in 1999, and the latest version is HTML 5. We can save HTML files with an extension .html.

Markup Language is a language that is interpreted by the browser and it defines the elements within a document using "tags". It is human-readable, which means that markup files use common words rather than the complicated syntax of programming languages.
HTML Example

```
<!DOCTYPE html>
<html>
<head>
<title>First HTML Code</title>
</head>
<body>
<h2>Welcome To GFG</h2>
<p>Hello Geeks</p>
</body>
</html>
```

3.3 HTML Elements and Tags

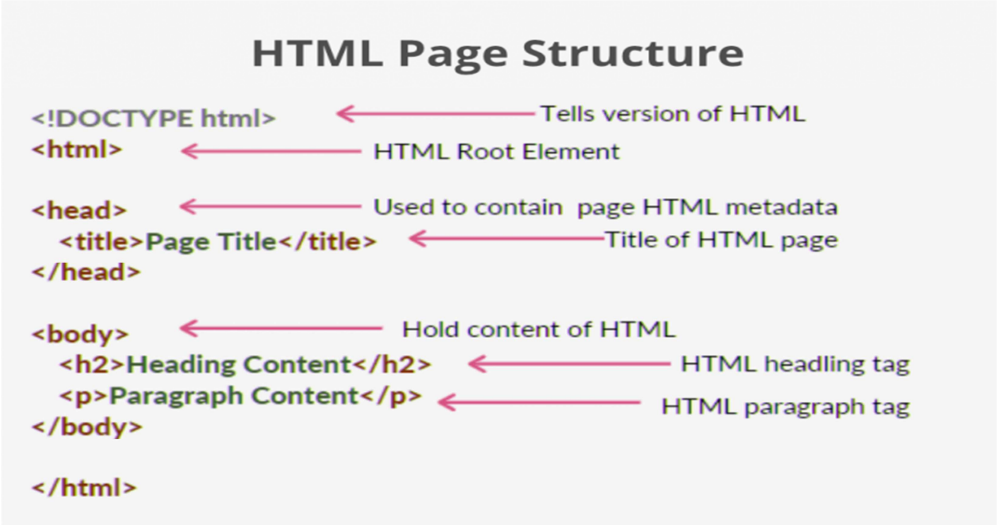
HTML uses predefined tags and elements that instruct the browser on how to display the content. HTML elements include an opening tag,

some content, and a closing tag. It's important to remember to include closing tags. If omitted, the browser applies the effect of the opening tag until the end of the page.

This section will dive into the basic structure of an HTML page, which includes essential building-block elements like doctype declaration, HTML, head, title, and body elements.

HTML Page Structure

The basic structure of an HTML page is shown below. It contains the essential building-block elements (i.e. doctype declaration, HTML, head, title, and body elements) upon which all web pages are created.



<!DOCTYPE html> – This is the document type declaration (not technically a tag). It declares a document as being an HTML document. The doctype declaration is not case-sensitive.

<html> – This is called the HTML root element. All other elements are contained within it.

<head> – The head tag contains the "behind the scenes" elements for a webpage. Elements within the head aren't visible on the front end of a webpage. HTML elements used inside the <head> element include:

<style> – This HTML tag allows us to insert styling into our web pages and make them appealing to look at with the help of CSS.
<title> – The title is what is displayed on the top of your browser when you visit a website and contains the title of the webpage that you are viewing.
<base> – It specifies the base URL for all relative URL's in a document.
<noscript> – Defines a section of HTML that is inserted when the scripting has been turned off in the user's browser.
<script> – This tag is used to add functionality to the website with the help of JavaScript.
<meta> – This tag encloses the metadata of the website that must be loaded every time the website is visited. For eg:- the metadata charset allows you to use the standard UTF-8 encoding on your website. This in turn allows the users to view your webpage in the language of their choice. It is a self-closing tag.
<link> – The 'link' tag is used to tie together HTML, CSS, and JavaScript. It is self-closing.
<body> – The body tag is used to enclose all the visible content of a webpage. In other words, the body content is what the browser will show on the front end.
An HTML document can be created using an HTML text editor. Save the text file using the ".html" or ".htm" extension. Once saved as an HTML document, the file can be opened as a webpage in the browser.
Note: Basic/built-in text editors are Notepad (Windows) and TextEdit (MacOS). Other advanced text editors include Sublime Text, Visual Studio Code, Froala, etc.
This example illustrates the basic structure of HTML code.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
<!--The above meta characteristics make a website
 compatible with different devices. -->
<title>Demo Web Page</title>
</head>
<body>
<h1>IFT 2023 Class</h1>
<p>A computer science portal for geeks</p>
</body>
</html>
```

Doctype Declaration:
The doctype declaration (<!DOCTYPE html>) is placed at the very beginning of an HTML document. It informs the browser about the

version of HTML being used, which is HTML5 in this case. This declaration ensures that the document is rendered correctly.

HTML Tag:
The <html> tag serves as the root element of an HTML document. It encompasses the entire content of the page and provides the base for all other elements.

Head Section:
The <head> section comes after the opening <html> tag and before the <body> tag. It contains meta-information about the document, such as the page title, character encoding, CSS stylesheets, JavaScript files, and other metadata.

The content within the <head> section is not directly visible on the page but is crucial for search engines and browsers to understand and process the document correctly.

Title Tag:
The title tag is placed within the <head> section. It defines the title of the web page, which is displayed on the browser's title bar or in search engine results. The text enclosed within the <title> tags is a concise description of the page's content and should accurately represent its purpose.

Body Section:
The <body> tag contains the visible content of the web page. It includes text, images, links, and interactive elements like forms or buttons. Everything between the opening <body> tag and the closing </body> tag is considered the body content.

Here is a basic HTML template that illustrates the standard structure: html

```
<!DOCTYPE html>
<html>
<head><title>Page Title</title>
</head>
<body>
<h1>Welcome to My Website</h1>
<p>This is the content of my web page.</p>
<a href="https://www.testing.com">Visit Example</a>
<!-- Additional elements and content go here →
</body>
</html>
```

In the example above, the <title> tag sets the page title, and within the <body> section, we have a heading <h1>, a paragraph , and a link <a> to the website "www.testing.com". This template is a starting point, and you can add more elements and content within the <body> section to build your web page.

Remember, this is a basic representation of the HTML structure. As you progress in web development, you will encounter more complex elements and tags to enhance the functionality and design of your web pages.

3.4 Essential HTML Tags

Heading Tags
Heading tags, from <h1> to <h6>, are used to define headings and subheadings in a web page. They play a crucial role in organizing and structuring content. The higher the number in the heading tag, the less significant the heading is. Here is an example:

html
<h1>This is the Main Heading</h1>
<h2>This is a Subheading</h2>
Heading tags not only provide visual hierarchy but also contribute to search engine optimization (SEO) by signalling the importance of content to search engines
Paragraph and Break Tags
The tag is used to define paragraphs of text. It is a block-level element that creates a new paragraph. Here is an example:

html
This is a paragraph of text.
Another paragraph of text.
If you want to create a line break within a paragraph, you can use the <br> tag, which is an empty element. Here is an example:

html
This is the first line.
This is the second line.
Anchor Tags
The <a> tag is used to create hyperlinks. It allows you to link to other web pages, documents, or specific parts of a page. The href attribute specifies the URL of the destination. Here is an example:

html
Visit Example

You can also use the target attribute to control how the link opens. For example, target="_blank" opens the link in a new browser tab:

html
Visit Example
Image Tags

The <img> tag is used to embed images in a web page. The src attribute specifies the image source (file URL or path). The alt attribute provides alternative text that describes the image for accessibility purposes. **Here is an example:**

html
<img src="image.jpg" alt="Description of the image" title="Image Title">
You can also use the title attribute to provide additional information about the image when the user hovers over it: html
<img src="image.jpg" alt="Image Description" title="Additional Information">

List Tags
HTML provides two types of lists: unordered lists (<ul>) and ordered lists (<ol>). The <li> tag is used to define individual list items within these lists. Here is an example:

html

Item 1
Item 2

First
Second

Table Tags
Tables are used to present data in a structured format. The <table> tag creates the table, while the <tr> tag defines table rows. Within each row, we use the <td> tag to define table cells. The <th> tag is used for table headers. Here is an example:

html
<table>
<tr>
<th>Header 1</th>

```
<th>Header 2</th>
</tr>
<tr>
<td>Data 1</td>
<td>Data 2</td>
</tr>
</table>
```

Div and Span Tags

The tag is used for grouping elements together and applying styles or JavaScript to them. The tag is used for inline styling or targeting specific parts of the text. Here is an example:

html

```
<div><p>This is a group of elements.</p>
<span style="color: blue;">This text is styled differently.</span></div>
```

Form Tags

The <form> tag is used to create a web form. The <input> tag represents various input types such as text fields, checkboxes, radio buttons, etc. The <textarea> tag is used for multi-line text input, and the <button> tag represents a clickable button. Here is an example.

html

```
<form action="/submit-form" method="POST">
<label for="name">Name:</label>
<input type="text" id="name" name="name" required>
<br>
<label for="email">Email:</label>
<input type="email" id="email" name="email" required>
<br>
<textarea id="message" name="message" rows="4" cols="50" required></textarea><br>
<button type="submit">Submit</button>
</form>
```

Important HTML Tags

<!DOCTYPE html>: Defines the type of document. Here it defines that the document type is HTML.

<html></html>: It is the root element and all the other tags are contained in it. It determines the start and the end of the HTML document.

<head></head>: It contains metadata of the HTML document & is actually not displayed on the webpage. The heading starts with <head> and end with </head>.

<title></title>: It is used to create a title of the document and the title appears in the title bar at the top. At least one title appears in every document. The title portion of the document starts with <title> and ends with </title>, and in between, enter the text that you want as the title.
<body></body>: It contains the contents of the document to be displayed on the web page. The content may be an image, some text, some links, etc. This part represents the body of the web document, which often includes headings, text, and paragraphs.
<p>: It is used for defining a paragraph.

: It is used for a single-line break.
<img>: It is used for defining an image with a given source.
<sup>: It is used for defining superscripted data.
<b>: It is used for defining bold text.
<sub>: It is used for defining subscripted data, etc.
Example 1: Save the following by MyGeeksHtml.html.

```
<!DOCTYPE html>
<html>
<head>
        <title> IFT 203 class webpage</title>
</head>
<body>
        My First WebPage
<p> I am using paragraph tag </p>
        Now i am out of paragraph tag
        No line break
        <br> I am using line break tag
        <b> Now using Bold Tag </b>
        <img src=
"https://media.IFT 203 Class.org/wp-content/cdn-uploads/gfg_200x200-min.png"
                width="200" height="100">
</body>
</html>
```

Example 2: In this example, we will use all the heading tags from <h1> to <h6>

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport"
                content="width=device-width, initial-scale=1.0">
        <title>HTML</title>
</head>
<body>
```

```
<h1>Welcome IFT 203</h1>
<h2> Welcome IFT 203</h2>
<h3> Welcome IFT 203</h3>
<h4> Welcome IFT 203</h4>
<h5> Welcome IFT 203</h5>
<h6> Welcome IFT 203</h6>
</body>
</html>
```

3.5 Importance of HTML

HTML is essential to the internet as it provides the structure, formatting, and functionality required to create web pages. Understanding HTML is the first step in web development, enabling individuals to build functional and visually appealing websites.

Structure and Organization
HTML provides a logical structure for web content. It allows developers to define headings, paragraphs, lists, and other elements, which not only enhance readability but also aid in search engine optimization (SEO).

Cross-Platform Compatibility
HTML ensures cross-platform compatibility, making web pages accessible across different devices and operating systems. It allows content to adapt to various screen sizes and resolutions, enabling responsive web design and ensuring a consistent user experience.

Hyperlinks and Navigation
HTML enables hyperlink creation for seamless navigation between web pages. Hyperlinks are vital for the interconnected nature of the internet allowing easy movement within and across websites. HTML's <a> tag makes it possible to link to external resources, internal pages, or specific page sections.

Media Integration
HTML supports the inclusion of multimedia elements, such as images, audio, and video, within web pages. By using HTML's <img>, <audio>, and <video> tags, developers can embed visual and auditory content directly into their websites, enhancing engagement and user experience.

Forms and User Interaction
HTML forms enable user interaction and data collection on websites. Forms allow users to input data, submit information, and interact with web applications. HTML provides elements like text fields,

checkboxes, radio buttons, dropdown menus, and submit buttons, which are essential for creating interactive web pages.

Foundation for Web Technologies
HTML is the foundation for web technologies like CSS (Cascading Style Sheets) and JavaScript. CSS styles HTML elements, improving the visual presentation and layout of web pages, while JavaScript adds interactivity and dynamic functionality to HTML-based websites.

3.6 Best Practices in HTML Coding

Following coding standards is essential for clean, maintainable, and efficient HTML code. Consistent and well-structured code improves readability, makes collaboration easier, and minimizes errors. Some best practices to consider include:
Indentation and Formatting: Use consistent indentation and proper formatting to enhance code readability. This helps you and other developers understand the structure of the HTML document more easily.

Naming Conventions: Use descriptive and meaningful names for HTML elements, classes, and IDs. This improves code clarity and makes it easier to understand the purpose of each element.

Avoid Inline Styles: Inline styles should be used sparingly. Instead, consider using external or internal CSS to separate styling from HTML structure, promoting better organization and maintainability.

Semantic HTML: Utilize semantic HTML elements (e.g., <header>, <nav>, <section>, <article>, <footer>) to convey the structure and meaning of the content. Semantic HTML improves accessibility, search engine optimization, and overall code quality.

Consistent Tag Naming: Stick to lowercase tag names to ensure consistency and compatibility across different browsers and versions.

Self-Assessment Exercise(s)

(1) What does HTML stand for?
A) Hyperlinks and Text Markup Language
B) Home Tool Markup Language
C) Hyper Text Markup Language
D) Hyper Tool Multi-Language
Answer: C) Hyper Text Markup Language

(2) Which HTML tag is used to create a hyperlink?

A) <a>
B) <link>
C) <href>
D) <hyperlink>
Answer: A) <a>

(3) Which of the following is a block-level element in HTML?
A)
B)
C) <a>
D) <img>
Answer: B)

(4) Which attribute is used to specify an alternate text for an image, if the image cannot be displayed?
A) alt
B) title
C) src
D) href
Answer: A) alt

What is the purpose of the <head> tag in an HTML document?
A) To contain the main content of the page
B) To link external files
C) To contain meta-information about the document
D) To display images and links
Answer: To contain meta-information about the document

Conclusion

Understanding the fundamentals of HTML is essential for anyone looking to develop or engage with web technologies. HTML forms the backbone of web content, providing the structure and organization necessary to present text, images, links, multimedia, and interactive forms on the web. With its elements and tags, HTML allows developers to create structured and semantically meaningful web pages that are easily interpreted by browsers and accessible to users across various devices and platforms. The introduction of HTML5 has further enhanced the language's capabilities, adding new elements and APIs to support modern web applications. Mastering HTML is a foundational step in web development, enabling the creation of rich, user-friendly, and dynamic web experiences.

4.0 Summary

In today's digital age, having a basic understanding is essential for anyone aspiring to create a website. HTML forms the foundation of every web page, allowing developers to structure content and create interactive websites. In this unit, we introduce fundamental HTML codes to help you start building your website. We cover document structure, headings, paragraphs, links, images, lists, tables, and forms. Whether you are a beginner or want to refresh your HTML skills, this unit will be a valuable guide.

5.0 References/Further Reading

Robbins, J. N. (2012). Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. " O'Reilly Media, Inc.".

Mercer, D. (2001). Schaum's Outline of HTML. McGraw-Hill, Inc..

Cook, C., & Schultz, D. (2007). Beginning HTML with CSS and XHTML: Modern Guide and Reference. Apress.

Macaulay, M. (2017). Introduction to web interaction design: With Html and Css. Chapman and Hall/CRC.

McGrath, M. (2020). HTML, CSS & JavaScript in easy steps. In Easy Steps Limited.

Tittel, E., & Noble, J. (2010). HTML, XHTML and CSS for dummies. John Wiley & Sons.

Powell, T. (2010). HTML & CSS: the complete reference. McGraw-Hill, Inc.

Unit 2 HTML Tags and Attribute

Contents

1.0 Introduction

HTML tags are the basic building blocks of HTML, used to create and structure sections, paragraphs, and links on a web page. An HTML tag consists of an element name enclosed in angle brackets, such as for a paragraph or <a> for a hyperlink. Tags often come in pairs, with an opening tag ( ) and a closing tag ( ), where the closing tag includes a forward slash. Some tags, like <img> for images and <br> for line breaks, are self-closing and do not require a closing tag. HTML tags help browsers understand how to display the content, whether it's text, images, or other media, ensuring that the web page appears as intended by the developer.

HTML attributes provide additional information about an element and are included within the opening tag. Attributes typically come in name-value pairs, separated by an equals sign, and are enclosed in quotation marks, such as class="classname" or src="image.jpg". Common attributes include id for unique identification, class for assigning CSS styles, href for specifying the destination of a link, and alt for providing alternative text for images. Attributes enhance the functionality and accessibility of HTML elements, allowing for more precise control over the web page's behavior and presentation. For example, using the style attribute can directly apply CSS styles to an element, while the data-* attributes can store custom data private to the page or application.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, I will be able to:

- discuss the meaning of HTML Tags

- use simple HTML Tags codes
- use simple HTML tags in practical designs
- explain HTML Attributes

3.0 Main Content

3.1 The HTML Tags

HTML (HyperText Markup Language) is the standard language used to create and design webpages. It uses a series of elements, represented by tags, to structure and format content on the web. Tags are enclosed in angle brackets, for example, <tagname>. Each tag usually comes in a pair: an opening tag <tagname> and a closing tag </tagname>, although some tags are self-closing.

The basic structure of an HTML document begins with the <!DOCTYPE html> declaration, followed by the <html> tag, which wraps all the content on the page. Within the <html> tag, there are two main sections: <head> and <body>. The <head> section contains meta-information about the document, such as the title (specified using the <title> tag) and links to stylesheets or scripts. For instance, <head> might include <title>My Webpage</title>. The <body> section contains the actual content that is displayed on the webpage, such as text, images, and links.

Text content in HTML is structured using various tags. Paragraphs are created with the tag, like This is a paragraph. . Headings are defined with tags <h1> through <h6>, with <h1> being the highest level of heading and <h6> the lowest, for example, <h1>Main Heading</h1>. To emphasize text, the and tags are used, which typically render text in bold and italic, respectively: Important and Italicized.

Links and images are also integral parts of HTML. Hyperlinks are created using the <a> tag, where the href attribute specifies the URL: Visit Example. Images are embedded with the <img> tag, which is self-closing and requires the src attribute to specify the image source, and optionally, the alt attribute for alternative text: <img src="image.jpg" alt="Description of image">. Lists are another common element, which can be ordered <ol> or unordered <ul>, with list items defined by the <li> tag: First item
Second item.

These examples showcase the fundamental building blocks of HTML. By combining these tags and understanding their attributes and usage, developers can create well-structured and visually appealing webpages. As web technology evolves, HTML remains a critical skill, enabling the creation and maintenance of dynamic, interactive, and accessible web content.

HTML consists of standardized “tags” that are used to define the structure of information on Web pages. The decision about the structure of the text is made by the browser based on the tags, which are marks that are embedded into the text. A tag is enclosed in two signs (< and >) and usually comes in pairs. The beginning tag starts with the name of the tag, and the ending tag starts with a slash followed by the name of the tag. The use of tags enables web pages to have many features including bold text, italic text, heading, paragraph breaks, and numbered or bulleted list. The table shows a list of common HTML tags

Opening Tag	**Closing Tag**	**Meaning**
<A>	</A>	Defines an address (hyperlink)
<BODY>	</BODY>	Defines the body of the document
 		Line break
<HEAD>	</HEAD>	Defines the head of the document
<HTML>	</HTML>	Defines an HTML document
<IMG>		Define an Image
<LI>	</LI>	An item in a list
<OL>	</OL>	Ordered list
<UL>	<//UL>	Unordered list
<TITLE>	</TITLE>	Defines the title of the document

Tags are generally used to specify “mark-up” regions of HTML documents for the web browser to interpret. Tags are composed of the name of the element, surrounded by angle brackets. An end tag also has a slash after the opening angle bracket, to distinguish it from the start tag. For example, p, which represents a paragraph by p element, would be written as:

 This represent a paragraph

Not all elements require the end tag. An example of an element that does not require an end tag is the <br> element which forces a line break on the display of interpreted HTML codes on a browser.

HTML attributes are modifiers of HTML elements. They generally appear as name-value pairs, separated by "=", and are written within the start tag of an element, after the element's name:

<"tag" "attribute""value"">(content to be modified by the tag)</tag>

Where the tag names the HTML element, an attribute is the name of the attribute, set to the provided value. An attribute customizes or modifies HTML elements.

3.2 The structure of HTML Page

The basic structure for all HTML documents is simple and should include the following minimum elements or tags:

<html>-This is the starting tag of the html which must be closed at the end of the page

<head>-The author of the page can insert his/her details here</head>

<title>-The is used for the title of the page which is published on the title page of your web browser</title>

<body>-This is a container of the main body of the page</body>

</html>

The <HTML> Element

The HTML element is considered the root and container element for the whole HTML document. That is, its sole purpose is to encapsulate all the HTML code and describe the HTML document to the web browser. Each HTML document should have one <html> and each document should end with a closing </html> tag.

Example 1: HTML Code:

<html>

…..the contents should be here in the order of the head, title and body

</html>

The <HEAD> Element

The HEAD tag marks the beginning of the document head element; its contains the title of the pages and other parameters that the browser will use. Thus, each <head> element should contain a <title> element indicating the title of the document, and may also contain any combination of the following elements, in any order:

The <style> tag.

This is used for declaring or including Cascading Style Sheets(CSS) codes inside your HTML document.

The <script> tag

This tag is used to declare or include JAVAScript or VBScript inside the document.

The <meta> tag

This is used to include information about the document such as keywords and a description, which are particularly helpful for search applications.

The <base> tag

This is used to create a "base" universal resource location (url) for all links on the page.

The <object> tag

This is designed to include multimedia such as images, Flash animations, MP3 files, QuickTime movies JavaScript objects, and other components of a page. The <param> tag is used along with this

tag to define various parameters. Note the <embed> tag can also be used to include multimedia files as will be discuss later in this module.

The <link> tag

This is used to link to an external file, such as a style sheet or JavaScript file.

Example 2: Codes for HEAD element

```
<head>
<meta name="Keywords" content="NOUN, Web Pages" /><meta name="description" content="HTML Basic Tags" />
        <base href="http://www.noun.edu.nghttp://www.nou.edu.ng/ "
/>
<link rel="stylesheet" type="text/css" href="noun.css" /><script type="text/ javascript"> _uacct = "UA -232293"; urchinTracker();
< /script>
</head>
```

The <title> Element

The <title> tag is usually placed within the <head> element to title your page. Whatever is written between the opening and closing <title></title> tags will be displayed in the title bar of the WEB browser. Search engines that use its content to help index pages use the title information. Therefore, it is excellent practice to use a title that really describes the content of your site.

Example 3: Code for Title element

```
< html>
<head>
<title>National Open University of Nigeria </title>
</head>
</html>
```

The <Body> Element

The <body> element appears after the <head> element. The purpose of the <body> element is to contain the text and HTML element that will display in the browser window. A <body> element may contain anything from a couple of paragraphs, links, images under a heading to more complicated layouts containing forms and tables. We will be looking at each of these elements in detail later in this unit. For now, it is only important to understand that the body element will encapsulate all of your webpage viewable content.

Example 4: Codes for Body Element

```
<html>
< head>
<title>National Open University Website!</title></head>
< body>
Welcome to the official Website of the National Open University of Nigeria </body>
</html>
```

3.2 The HTML attributes

HTML attributes provide additional information about HTML elements. They are always specified in the start tag of an element and usually come in name/value pairs like `name="value"`. Attributes can define properties such as the element's id, class, style, or behavior. These properties influence how elements are presented or interact on a webpage. For example, in `<a href="https://www.example.com">Visit Example</a>`, the `href` attribute specifies the URL that the hyperlink points to.

The `class` attribute is commonly used to apply CSS styles to elements. It allows you to group multiple elements together and apply the same styles to them. For instance, `<div class="container">Content here</div>` uses the `class` attribute to assign the class `container` to a `div` element. This class can then be targeted in a CSS file to apply specific styles, such as `background-color: lightblue;` to all elements with the class `container`.

The `id` attribute is used to uniquely identify an element within a document. It is especially useful for targeting elements with CSS and JavaScript. For example, `<p id="intro">This is an introduction paragraph.</p>` assigns the id `intro` to a paragraph. In CSS, you can style this specific paragraph with `#intro { font-weight: bold; }`. In JavaScript, you can manipulate it using `document.getElementById("intro").style.color = "red";`.

Another important attribute is the `alt` attribute, used with the `img` tag to provide alternative text for images. This text is displayed if the image cannot be loaded and is also used by screen readers to describe images for visually impaired users. For example, `<img src="photo.jpg" alt="A beautiful scenery of mountains and lakes">` ensures that users understand the context of the image even if it fails to load. This not only improves accessibility but also helps with search engine optimization by providing descriptive text for images.

HTML attributes are essential components of HTML elements, providing additional information that defines their properties, behavior, and overall presentation. These attributes are always specified in the opening tag of an HTML element and are written in name/value pairs, like `name="value"`. Understanding and effectively using HTML attributes is crucial for web development, as they help create rich, interactive, and accessible web pages. This detailed explanation will cover various types of HTML attributes, their uses, and examples to illustrate their application.

Basic HTML Attributes
`id` Attribute
The `id` attribute is used to uniquely identify an HTML element within a document. This uniqueness is critical because it allows developers to target specific elements with CSS or JavaScript. For instance, in CSS, you can style an element with a particular `id`, and in JavaScript, you can manipulate it directly.
Example:
```html
<p id="intro">This is an introduction paragraph.</p>
```
In CSS, you can target this paragraph:
```css
#intro {
    font-weight: bold;
    color: blue;
}
```
In JavaScript, you can change its properties:
```javascript
document.getElementById("intro").style.color = "red";
```
`class` Attribute
The `class` attribute is used to apply styles and behaviors to multiple elements. Unlike the `id` attribute, a class can be reused on multiple elements, making it very useful for styling groups of elements.
Example:
```html
<div class="container">This is a container.</div>
<div class="container">This is another container.</div>
```
In CSS, you can style all elements with the class `container`:
```css
.container {
    padding: 20px;
    background-color: lightgrey;
}
```
`style` Attribute
The `style` attribute allows you to apply inline CSS to an element. This can be useful for quick styling, but it's generally recommended to use external or internal CSS for maintainability.
Example:
```html
<p style="color: green; font-size: 16px;">This is a styled paragraph.</p>
```

```
#### `title` Attribute
The `title` attribute provides additional information about an element, often displayed as a tooltip when the mouse hovers over it.
Example:
```html
<p title="This is a tooltip">Hover over this text to see the tooltip.</p>
```
Form Attributes
Forms are essential for collecting user input on web pages. Several attributes are specifically designed to enhance form functionality.
`action` and `method` Attributes
The `action` attribute specifies where to send the form data when the form is submitted, and the `method` attribute defines how to send the data (usually `GET` or `POST`).
Example:
```html
<form action="/submit-form" method="post">
<input type="text" name="username">
<input type="submit" value="Submit">
</form>
```
`placeholder` Attribute
The `placeholder` attribute provides a hint to the user of what can be entered in the input field.
Example:
```html
<input type="text" placeholder="Enter your name">
```
`required` Attribute
The `required` attribute specifies that an input field must be filled out before submitting the form.
Example:
```html
<input type="email" required>
```
Global Attributes
Global attributes can be applied to any HTML element, enhancing its functionality and accessibility.
`data-*` Attributes
Custom data attributes, or `data-*` attributes, allow you to store extra information on standard HTML elements. This information can be used in JavaScript.
Example:
```html
<div data-user-id="12345" data-role="admin">User Info</div>
```
```

```
In JavaScript, you can access these attributes:
```javascript
let userId = document.querySelector('[data-user-id]').getAttribute('data-user-id');
```
`hidden` Attribute
The `hidden` attribute is used to hide an element from the user. It can be toggled with JavaScript to show or hide elements dynamically.
Example:
```html
<p hidden>This paragraph is hidden.</p>
```
Image Attributes
Images are integral to web content, and specific attributes help manage their display and accessibility.
`src` Attribute
The `src` attribute specifies the path to the image file.
Example:
```html
<img src="image.jpg" alt="Description of image">
```
`alt` Attribute
The `alt` attribute provides alternative text for an image, which is crucial for accessibility and SEO. This text is displayed if the image cannot be loaded and is read by screen readers.
Example:
```html
<img src="photo.jpg" alt="A beautiful scenery of mountains and lakes">
```
Link Attributes
Links connect web pages and resources, and specific attributes enhance their functionality.
`href` Attribute
The `href` attribute specifies the URL of the page the link goes to.
Example:
```html
<a href="https://www.example.com">Visit Example</a>
```
`target` Attribute
The `target` attribute specifies where to open the linked document. For example, `_blank` opens the link in a new tab.
Example:
```html

<a href="https://www.example.com" target="_blank">Visit Example in new tab</a>
```
Table Attributes
Tables organize data into rows and columns, and specific attributes manage their structure and presentation.
`colspan` and `rowspan` Attributes
The `colspan` attribute allows a cell to span multiple columns, while `rowspan` allows a cell to span multiple rows.
Example:
```html
<table>
<tr>
<td colspan="2">This cell spans two columns</td>
</tr>
<tr>
<td rowspan="2">This cell spans two rows</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 2</td>
</tr>
</table>
```
Event Attributes
Event attributes allow you to specify JavaScript code to execute in response to events.
`onclick` Attribute
The `onclick` attribute executes JavaScript when an element is clicked.
Example:
```html
<button onclick="alert('Button clicked!')">Click me</button>
```
Accessibility Attributes (ARIA)
ARIA (Accessible Rich Internet Applications) attributes help improve accessibility for users with disabilities.
`aria-label` Attribute
The `aria-label` attribute provides an accessible name for an element that can be read by screen readers.
Example:
```html
<button aria-label="Close">X</button>
```
`role` Attribute
The `role` attribute defines the role of an element in a web application, making it more accessible.
```

Example:
```html
<div role="navigation">Navigation Menu</div>
```
Custom Attributes
HTML5 allows for the use of custom data attributes to store additional information about elements.
`data-*` Attributes
Custom data attributes are prefixed with `data-` and allow developers to embed custom data attributes in elements. These attributes are useful for storing information that can be manipulated via JavaScript.

Example:
```html
<div data-product-id="12345" data-category="books">Product Information</div>
```
Examples of Complex Use
Combining multiple attributes in a single element can create powerful and interactive web elements.
Example:
```html
<a href="https://www.example.com" target="_blank" id="exampleLink" class="link-class" title="Example Site" data-info="additional data" onclick="trackClick(this)">Visit Example</a>
```
In this example, the anchor tag uses several attributes:
- `href` to define the URL
- `target` to open the link in a new tab
- `id` to uniquely identify the link
- `class` to apply CSS styles
- `title` to provide a tooltip
- `data-info` to store custom data
- `onclick` to execute JavaScript when the link is clicked

Tags vs Elements vs Attributes difference

HTML Tags	HTML Elements	HTML Attributes
HTML tags are used to hold the HTML element.	HTML element holds the content.	HTML attributes are used to describe the characteristics of an HTML element in detail.
HTML tag starts with < and ends	Whatever written within a HTML tag	HTML attributes are found only in the starting

HTML Tags	HTML Elements	HTML Attributes
with >	are HTML elements.	tag.
HTML tags are almost like keywords where every single tag has unique meaning.	HTML elements specifies the general content.	HTML attributes specify various additional properties to the existing HTML element.
Tags define the type of HTML element (e.g., heading, paragraph).	Elements represent the complete, functional unit on a webpage.	Attributes provide extra information or settings for elements.

Self-Assessment Exercise(s)

1) Which HTML tag is used to define a hyperlink?
a) <link>
b) <a>
c) <h>
d) <p>
Answers: b) <a>

(2) Which attribute is used to specify the URL of the image to be displayed in an <img> tag?
a) src
b) href
c) alt
d) title
Answers: a) src

(3) Which tag is used to define the structure of an HTML document, including the title and metadata?
a) <header>
b) <body>
c) <head>
d) <title>
Answers: c) <head>

(4) Which attribute is used to specify an alternate text for an image in case the image cannot be displayed?
a) alt
b) src
c) title
d) href
Answers: a) alt

(5) Which tag is used to create a line break in HTML?
a)
b) <hr>
c) <p>
d) <nl>
Answers: a)

Conclusion

Understanding HTML tags and attributes is fundamental to web development, as they form the backbone of any webpage. HTML tags, such as , <p>, and <a>, define the structure and content of a webpage, while attributes like class, id, and href provide additional information and functionality to these tags. By learning how to use these elements effectively, developers can create well-structured, accessible, and visually appealing websites. Mastery of HTML tags and attributes also paves the way for integrating more advanced web technologies such as CSS and JavaScript, enhancing the user experience and interactivity. A solid grasp of HTML tags and attributes is crucial for anyone looking to build or manage web content. These elements not only dictate how information is displayed but also influence how users interact with a website. As the foundation of web development, HTML ensures that content is properly formatted and accessible across different devices and browsers. Continual practice and staying updated with the latest HTML standards will help developers maintain the relevancy and efficiency of their web projects, ultimately contributing to the dynamic and ever-evolving nature of the internet.

4.0 Summary

The HTML tags and attributes unit covers the fundamental building blocks of web development, focusing on the essential elements that structure web content. HTML uses tags to create elements such as headings, paragraphs, links, images, and lists. Each tag is enclosed in angle brackets, with opening and closing tags defining the start and end

of an element, respectively. For example, <p> is used for paragraphs, and <a> is used for hyperlinks. The unit also explains the hierarchical nature of HTML, where elements can be nested within other elements to create a structured and organized web page layout.

In addition to tags, HTML attributes are crucial for providing additional information about elements. Attributes are included within the opening tag and typically come in name-value pairs, such as id="uniqueID" or class="classname". These attributes can control the behavior, appearance, and identity of HTML elements. For instance, the href attribute in an <a> tag specifies the URL of the link, while the src attribute in an <img> tag defines the image source. The unit emphasizes the importance of properly using tags and attributes to create semantically meaningful and accessible web pages, ensuring that content is both user-friendly and optimized for search engines.

5.0 References/Further Reading

Robbins, J. N. (2012). Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. " O'Reilly Media, Inc.".

Mercer, D. (2001). Schaum's Outline of HTML. McGraw-Hill, Inc..

Cook, C., & Schultz, D. (2007). Beginning HTML with CSS and XHTML: Modern Guide and Reference. Apress.

Macaulay, M. (2017). Introduction to web interaction design: With Html and Css. Chapman and Hall/CRC.

McGrath, M. (2020). HTML, CSS & JavaScript in easy steps. In Easy Steps Limited.

Powell, T. (2010). HTML & CSS: the complete reference. McGraw-Hill, Inc.

Unit 3 HTML syntax and basic markup: headings, paragraphs, lists, links

Contents

1.0 Introduction

HTML is the standard language used to create web pages. It consists of a series of elements represented by tags, which are enclosed in angle brackets. These tags generally come in pairs: an opening tag (e.g.,) and a closing tag (e.g.,). The content between these tags is what gets displayed on the web page. HTML documents start with a <!DOCTYPE html> declaration to define the document type and version, followed by an <html> element that encapsulates the entire content. Inside the <html> tag, there are two main sections: the <head>, which contains meta-information such as the title and links to stylesheets, and the <body>, which contains the actual content of the web page.

Basic HTML markup includes several fundamental elements. Headings are created using the <h1> to <h6> tags, with <h1> being the highest level and <h6> the lowest. Paragraphs are defined with the tag. Lists can be either ordered (numbered) or unordered (bulleted). Ordered lists use the <ol> tag, and each list item is enclosed in an <li> tag. Unordered lists use the <ul> tag, also with <li> for each item. Links are created using the <a> tag, which requires an href attribute to specify the URL. For example, Example creates a hyperlink to " https://noun.edu.ng " with the text "Example" as the clickable part. These basic elements provide the structure needed to build and organize content on web pages.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss the meaning of HTML Syntax
- explain HTML markup
- use simple HTML Markup Syntax
- use simple HTML Markup in practical designs

3.0 Main Content

3.1 HTML Syntax

HTML is the standard language used to create web pages. It structures the content on the web and is composed of elements represented by tags. Understanding HTML syntax and basic markup is essential for building web pages. This guide will cover the syntax and basic markup for headings, paragraphs, lists, and links with examples.

HTML documents are text files that contain HTML elements. An HTML element is defined by a start tag, content, and an end tag. The basic structure of an HTML document includes a `<!DOCTYPE html>` declaration, a `<html>` element, a `<head>` element, and a `<body>` element.

HTML is a standard markup language for web page creation. It allows the creation and structure of sections, paragraphs, and links using HTML elements (the building blocks of a web page) such as tags and attributes.

HTML has a lot of use cases, namely:
Web development. Developers use HTML code to design how a browser displays web page elements, such as text, hyperlinks, and media files.

Internet navigation. Users can easily navigate and insert links between related pages and websites as HTML is heavily used to embed hyperlinks.

Web documentation. HTML makes it possible to organize and format documents, similarly to Microsoft Word.

It's also worth noting that HTML is not considered a programming language as it can't create dynamic functionality, although it is now considered an official web standard. The World Wide Web Consortium (W3C) maintains and develops HTML specifications, along with providing regular updates.

The average website includes several different HTML pages. For instance, a home page, an about page, and a contact page would all have separate HTML files. HTML documents are files that end with a .html or .htm extension. A web browser reads the HTML file and renders its content so that internet users can view it.

All HTML pages have a series of HTML elements, consisting of a set of tags and attributes. HTML elements are the building blocks of a web page. A tag tells the web browser where an element begins and ends, whereas an attribute describes the characteristics of an element.

The three main parts of an element are:
Opening tag – used to state where an element starts to take effect. The tag is wrapped with opening and closing angle brackets. For example, use the start tag

 to create a paragraph.
Content – this is the output that other users see.
Closing tag – the same as the opening tag, but with a forward slash before the element name. For example,

 to end a paragraph.
The combination of these three parts will create an HTML element:
This is how you add a paragraph in HTML.
This is how you add a paragraph in HTML.
This is how you add a paragraph in HTML.
Another critical part of an HTML element is its attribute, which has two sections – a name and an attribute value. The name identifies the additional information that a user wants to add, while the attribute value gives further specifications.
For example, a style element adding the color purple and the font-family Verdana will look like this:
This is how you add a paragraph in HTML.
This is how you add a paragraph in HTML.
This is how you add a paragraph in HTML.

Another attribute, the HTML class, is most important for development and programming. The class attribute adds style information that can work on different elements with the same class value.
For example, we will use the same style for a heading
<h1>
<h1> and a paragraph

. The style includes background color, text color, border, margin, and padding, under the class .important. To achieve the same style between
<h1>
<h1> and

, add
class="important"
class="important" after each start tag:
Syntax Highlighter

```
<html>
<head>
<style>
.important {
background-color: blue;
color: white;
border: 2px solid black;
margin: 2px;
padding: 2px;
}
</style>
</head>
<body>
<h1 class="important">This is a heading</h1>
<p class="important">This is a paragraph.</p>
</body>
</html>
```

<html><head><style> .important { background-color: blue; color: white; border: 2px solid black; margin: 2px; padding: 2px; } </style></head><body><h1 class="important">This is a heading</h1> This is a paragraph. </body></html>

```
<html>
<head>
<style>
.important {
  background-color: blue;
  color: white;
  border: 2px solid black;
```

```
  margin: 2px;
  padding: 2px;
}
</style>
</head>
<body>

<h1 class="important">This is a heading</h1>
<p class="important">This is a paragraph.</p>

</body>
</html>
```

Most elements have an opening and a closing tag, but some elements do not need closing tags to work, such as empty elements. These elements do not use an end tag because they do not have content:

```
<img src="/" alt="Image">
<img src="/" alt="Image">
<img src="/" alt="Image">
```

This image tag has two attributes – an
src
src attribute, the image path, and an
alt
alt attribute, the descriptive text. However, it does not have content nor an end tag.

Lastly, every HTML document must start with a <!DOCTYPE> declaration to inform the web browser about the document type. With HTML5, the doctype HTML public declaration will be:

```
<!DOCTYPE html>
<!DOCTYPE html>
```

3.2 Basic HTML Document Structure

```html
<!DOCTYPE html>
<html>
<head>
<title>My First HTML Page</title>
</head>
<body>
<!-- Content goes here -->
</body>
</html>
```

`<!DOCTYPE html>`: This declaration defines the document type and version of HTML. In this case, it is HTML5.

- `<html>`: The root element of an HTML page.

- `<head>`: Contains meta-information about the HTML document, such as the title and links to stylesheets.
- `<title>`: Sets the title of the document, which appears in the browser's title bar or tab.
- `<body>`: Contains the content of the HTML document, such as text, images, links, and other elements.

Headings

Headings are used to define the structure and hierarchy of content. HTML provides six levels of headings, from `<h1>` to `<h6>`, with `<h1>` being the highest level and `<h6>` the lowest.

Examples of Headings

```html
<!DOCTYPE html>
<html>
<head>
<title>Headings Example</title>
</head>
<body>
<h1>Main Heading</h1>
<h2>Subheading</h2>
<h3>Sub-subheading</h3>
<h4>Level 4 Heading</h4>
<h5>Level 5 Heading</h5>
<h6>Level 6 Heading</h6>
</body>
</html>
```

`<h1>`: Used for the main title or the most important heading.
- `<h2>` to `<h6>`: Used for subheadings, with each subsequent number indicating a lower level of importance.

Paragraphs

Paragraphs are used to structure text content into blocks of text. The `<p>` tag is used to define a paragraph in HTML.

Example of a Paragraph

```html
<!DOCTYPE html>
<html>
<head>
<title>Paragraph Example</title>
</head>
<body>
<p>This is a paragraph of text. Paragraphs are used to group sentences together into blocks of content.</p>
</body>
```

```
</html>
```

- `<p>`: Defines a paragraph of text.

3.3 Lists

Lists are used to group related items. HTML provides two types of lists: ordered lists and unordered lists.

Ordered Lists
Ordered lists are used when the order of items matters. The `<ol>` tag defines an ordered list, and each item in the list is defined by an `<li>` tag.
Example of an Ordered List

```html
<!DOCTYPE html>
<html>
<head>
<title>Ordered List Example</title>
</head>
<body>
<ol>
<li>First item</li>
<li>Second item</li>
<li>Third item</li>
</ol>
</body>
</html>
```

- `<ol>`: Defines an ordered list.
- `<li>`: Defines a list item.

Unordered Lists
Unordered lists are used when the order of items does not matter. The `<ul>` tag defines an unordered list, and each item in the list is defined by an `<li>` tag.
Example of an Unordered List

```html
<!DOCTYPE html>
<html>
<head>
<title>Unordered List Example</title>
</head>
<body>
<ul>
<li>First item</li>
```

Second item
Third item

</body>
</html>
```
- `<ul>`: Defines an unordered list.
- `<li>`: Defines a list item.

**Nested Lists**
Lists can be nested within other lists to create sub-lists.
Example of a Nested List
```html
<!DOCTYPE html>
<html>
<head>
<title>Nested List Example</title>
</head>
<body>
<ul>
<li>First item
<ul>
<li>Sub-item 1</li>
<li>Sub-item 2</li>
</ul>
</li>
<li>Second item</li>
<li>Third item</li>
</ul>
</body>
</html>
```

3.4 Links

Links are used to navigate from one page to another or to different parts of the same page. The `<a>` tag defines a hyperlink, and the `href` attribute specifies the URL of the page the link goes to.
Example of a Link
```html
<!DOCTYPE html>
<html>
<head>
<title>Link Example</title>
</head>
<body>
<p>Visit the <a

href="https://www.example.com">Example</a>website.
</body>
</html>
```
- `<a>`: Defines a hyperlink.
- `href`: Specifies the URL of the linked page.

Internal Links
Internal links navigate to different sections within the same page. This is achieved using the `id` attribute.
Example of Internal Links
```html
<!DOCTYPE html>
<html>
<head>
<title>Internal Link Example</title>
</head>
<body>
<h2 id="section1">Section 1</h2>
<p>This is the first section of the document.</p>
<h2 id="section2">Section 2</h2>
<p>This is the second section of the document.</p>
<a href="#section1">Go to Section 1</a>
<a href="#section2">Go to Section 2</a>
</body>
</html>
```
- `id`: Defines a unique identifier for an element.
- `href="#id"`: Creates a link to an element with the specified `id`.

3.5 Combining Elements

HTML elements can be combined to create more complex structures and layouts.
Example Combining Headings, Paragraphs, Lists, and Links
```html
<!DOCTYPE html>
<html>
<head>
<title>Combined Elements Example</title>
</head>
<body>
<h1>Welcome to My Webpage</h1>
This is a paragraph introducing the content of the page.
<h2>Topics Covered</h2>
<ul>

```
<li>HTML Basics</li>
<li>CSS Styling
<ul>
<li>Selectors</li>
<li>Properties</li>
</ul>
</li>
<li>JavaScript Programming</li>
</ul>
<h2>Useful Links</h2>
<p>Here are some useful links:</p>
<ol>
<li><a href="https://www.w3schools.com">W3Schools</a></li>
<li><a href="https://developer.mozilla.org">MDN Web Docs</a></li>
</ol>
</body>
</html>
```

Self-Assessment Exercise(s)

(1) What is the correct HTML element for the largest heading?
A) <head>
B) <heading>
C) <h1>
D) <h6>
Answer: C) <h1>

(2) Which HTML element is used to define a paragraph?
A) <paragraph>
B) <para>
C) <p>
D) <pg>
Answer: C) <p>

(3) How do you create a hyperlink in HTML?
A) <link href="url">Link</link>
B) Link
C) Link
D) <hyperlink src="http://example.com">Link</hyperlink>
Answer: C) Link

(4) Which of the following elements is used to create an unordered list in HTML?
A)
B)
C) <list>

D) <uolist>
Answer: B)

(5) How do you add a background color in HTML?
A) <body bg="yellow">
B) <body background="yellow">
C) <body style="background-color:yellow;">
D) <body bgcolor="yellow">
Answer: C) <body style="background-color:yellow;">

Conclusion

Understanding HTML syntax and basic markup is fundamental for web development. This guide covered the basic structure of an HTML document and explained how to use headings, paragraphs, lists, and links. By mastering these elements, you can create well-structured and navigable web pages. Experiment with combining these elements to build more complex and dynamic content.

4.0 Summary

HTML is the standard language for creating web pages and web applications. HTML syntax consists of a series of elements that define the structure and content of a webpage. Each element is enclosed within angle brackets, with a start tag (e.g.,) and an end tag (e.g.,) that wraps the content. Some elements, like <img>, are self-closing and do not require an end tag. Tags can have attributes, which provide additional information about the element and are included within the start tag (e.g., <a href="https://example.com">). These attributes typically follow a key-value pair format.

The basic markup unit in HTML is the element. Elements can be nested inside one another to create a hierarchical structure that defines the layout and organization of the webpage. The most common elements include headings (<h1> to <h6>), paragraphs ( ), links (<a>), images (<img>), lists (<ul>, <ol>, and <li>), and divs ( ) for block-level content grouping. The document starts with a <!DOCTYPE html> declaration, followed by an <html> element that contains the <head> and <body> sections. The <head> element typically includes metadata, links to stylesheets, and scripts, while the <body> element contains the actual content that users interact with on the webpage.

5.0 References/Further Reading

Robbins, J. N. (2012). Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. " O'Reilly Media, Inc.".

Mercer, D. (2001). Schaum's Outline of HTML. McGraw-Hill, Inc..

Cook, C., & Schultz, D. (2007). Beginning HTML with CSS and XHTML: Modern Guide and Reference. Apress.

Macaulay, M. (2017). Introduction to web interaction design: With Html and Css. Chapman and Hall/CRC.

Tittel, E., & Noble, J. (2010). HTML, XHTML and CSS for dummies. John Wiley & Sons.

Unit 4 Advanced 5HTML and XHTML Elements

Contents

1.0 Introduction

HTML5, the latest version of the HyperText Markup Language, is a core technology of the World Wide Web. It standardizes how web content is structured and presented. Introduced to enhance multimedia capabilities without relying on additional plugins like Flash, HTML5 supports audio, video, and interactive elements natively. It also brings semantic elements such as <article>, <section>, and <nav>, which improve the readability of the code and the accessibility of web content. HTML5 is designed to be backward compatible, ensuring older web pages still function correctly, while introducing new features and improvements for modern web development.

Extensible HyperText Markup Language (XHTML) is a variant of HTML that combines the flexibility of HTML with the strict syntax rules of Extensible Markup Language (XML). Introduced to ensure that web documents are well-formed and can be parsed by XML parsers, XHTML requires that elements are properly nested, all tags are closed, and attributes are quoted. This strictness enhances the consistency and reliability of web documents across different platforms and browsers. However, because of its rigidity, XHTML can be less forgiving than HTML, leading to potential issues if the syntax rules are not meticulously followed. Despite its advantages, the adoption of HTML5 has overshadowed XHTML due to HTML5's more lenient syntax and broader feature set.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss the meaning of HTML5 and XHTLM Syntax
- explain HTML5 Attributes
- identify simple XHTML Elements
- use simple HTML5 and XHTML elements in practical designs

3.0 Main Content

3.1 The HTLM5

HTML5 is the latest major version of HTML, the standard language for creating and structuring web content. It was designed to improve the capabilities of the web by enhancing support for multimedia, interactive elements, and APIs, while ensuring compatibility across various devices and browsers. HTML5 introduces a variety of new elements and attributes that provide more semantic meaning, such as <header>, <footer>, <article>, <section>, and <nav>, which help developers structure their content more clearly and improve accessibility. Additionally, HTML5 removes many of the older, deprecated elements and attributes from previous versions, streamlining the development process.

One of the most significant advancements in HTML5 is its native support for multimedia elements, such as <video> and <audio>, which allow for the embedding of media files without requiring third-party plugins like Flash. This enhances the user experience by providing more reliable and efficient ways to display multimedia content. HTML5 also includes new APIs for drawing graphics (via the <canvas> element), offline web applications, local storage, and geolocation, which enable developers to create more dynamic and interactive web applications. These features, combined with improved error handling and better parsing rules, make HTML5 a powerful and flexible tool for modern web development.

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

Browser Support
The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

The mobile web browsers that come pre-installed on iPhones, iPads, and Android phones all have excellent support for HTML5.

New Features
HTML5 introduces several new elements and attributes that can help you in building modern websites. Here is a set of some of the most prominent features introduced in HTML5.

New Semantic Elements: These are like <header>, <footer>, and <section>.

Forms 2.0: Improvements to HTML web forms where new attributes have been introduced for <input> tag.
Persistent Local Storage − To achieve without resorting to third-party plugins.

WebSocket: A next-generation bidirectional communication technology for web applications.

Server-Sent Events: HTML5 introduces events that flow from a web server to the web browsers and they are called Server-Sent Events (SSE).

Canvas − This supports a two-dimensional drawing surface that you can program with JavaScript.

Audio & Video: You can embed audio or video on your webpages without resorting to third-party plugins.

Geolocation: Now visitors can choose to share their physical location with your web application.

Microdata: This lets you create your vocabularies beyond HTML5 and extend your web pages with custom semantics.

Drag and drop Drag and drop the items from one location to another location on the same webpage.

Backward Compatibility
HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers.

It is suggested to detect support for individual HTML5 features using a few lines of JavaScript.

3.2 HTML5 Syntax

The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web, but is not compatible with the more esoteric SGML features of HTML 4.
HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes, an attribute had to have a value and to close all empty elements.
HTML5 comes with a lot of flexibility and it supports the following features Uppercase tag names.
Quotes are optional for attributes.
Attribute values are optional.
Closing empty elements are optional.
The DOCTYPE
DOCTYPEs in older versions of HTML were longer because the HTML language was SGML based and therefore required a reference to a DTD.
HTML 5 authors would use simple syntax to specify DOCTYPE as follows −
<!DOCTYPE html>
The above syntax is case-insensitive.
Character Encoding
HTML 5 authors can use simple syntax to specify Character Encoding as follows −
<meta charset = "UTF-8">
The above syntax is case-insensitive.
The <script> tag
It's common practice to add a type attribute with a value of "text/javascript" to script elements as follows −
<script type = "text/javascript" src = "scriptfile.js"></script>

HTML 5 removes extra information required and you can use simply following syntax −
<script src = "scriptfile.js"></script>
The <link> tag
So far you were writing <link> as follows −
<link rel = "stylesheet" type = "text/css" href = "stylefile.css">
HTML 5 removes extra information required and you can simply use the following syntax −
<link rel = "stylesheet" href = "stylefile.css">
HTML5 Elements
HTML5 elements are marked up using start tags and end tags. Tags are delimited using angle brackets with the tag name in between.
The difference between start tags and end tags is that the latter includes a slash before the tag name.
Following is the example of an HTML5 element −
...
HTML5 tag names are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.
Most of the elements contain some content like ... contains a paragraph. Some elements, however, are forbidden from containing any content at all and these are known as void elements. For example, br, hr, link, meta, etc.
Here is a complete list of HTML5 Elements.

3.3 HTML5 Attributes

Elements may contain attributes that are used to set various properties of an element.
Some attributes are defined globally and can be used on any element, while others are defined for specific elements only. All attributes have a name and a value and look like as shown below in the example.
Following is the example of an HTML5 attribute which illustrates how to mark up a div element with an attribute named class using a value of "example" −
...
Attributes may only be specified within start tags and must never be used in end tags.
HTML5 attributes are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.
Here is a complete list of HTML5 Attributes.

HTML5 Document

The following tags have been introduced for better structure −

section − This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.

article − This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.

aside − This tag represents a piece of content that is only slightly related to the rest of the page.

header − This tag represents the header of a section.

footer − This tag represents a footer for a section and can contain information about the author, copyright information, et cetera.

nav − This tag represents a section of the document intended for navigation.

dialog − This tag can be used to mark up a conversation.

figure − This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

The markup for an HTML 5 document would look like the following −

```
<!DOCTYPE html>

<html>
<head>
<meta charset = "utf-8">
<title>...</title>
</head>

<body>
<header>...</header>
<nav>...</nav>

<article>
<section>
        ...
</section>
</article>
<aside>...</aside>

<footer>...</footer>
</body>
</html>
```

Live Demo

```
<!DOCTYPE html>

<html>
<head>
<meta charset = "utf-8">
```

```
<title>...</title>
</head>

<body>
<header role = "banner">
<h1>HTML5 Document Structure Example</h1>
<p>This page should be tried in safari, chrome or Mozila.</p>
</header>

<nav>
<ul>
<li><a href = "https://www.tutorialspoint.com/html">HTML
Tutorial</a></li>
<li><a href = "https://www.tutorialspoint.com/css">CSS
Tutorial</a></li>
<li><a href = "https://www.tutorialspoint.com/javascript">
      JavaScript Tutorial</a></li>
</ul>
</nav>

<article>
<section>
<p>Once article can have multiple sections</p>
</section>
</article>

<aside>
<p>This is  aside part of the web page</p>
</aside>

<footer>
<p>Created by <a href = "https://tutorialspoint.com/">Tutorials
Point</a></p>
</footer>

</body>
</html>
```

Some attributes are defined globally and can be used on any element, while others are defined for specific elements only. All attributes have a name and a value and look like as shown below in the example.

Following is the example of an HTML5 attributes which illustrates how to mark up a div element with an attribute named class using a value of "example" −

```
<div class = "example">...</div>
```

Attributes may only be specified within start tags and must never be used in end tags.

HTML5 attributes are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.

Standard Attributes

The attributes listed below are supported by almost all the HTML 5 tags.

Attribute	Options	Function
accesskey	User Defined	Specifies a keyboard shortcut to access an element.
Align	right, left, center	Horizontally aligns tags
background	URL	Places an background image behind an element
Bgcolor	numeric, hexidecimal, RGB values	Places a background color behind an element
Class	User Defined	Classifies an element for use with Cascading Style Sheets.
contenteditable	true, false	Specifies if the user can edit the element's content or not.
contextmenu	Menu id	Specifies the context menu for an element.
data-XXXX	User Defined	Custom attributes. Authors of a HTML document can define their own attributes. Must start with "data-".
Draggable	true,false, auto	Specifies whether or not a user is allowed to drag an element.
Height	Numeric Value	Specifies the height of tables, images, or table cells.
Hidden	Hidden	Specifies whether element should be visible or not.
Id	User Defined	Names an element for use with Cascading Style Sheets.
Item	List of elements	Used to group elements.

Itemprop	List of items	Used to group items.
spellcheck	true, false	Specifies if the element must have it's spelling or grammar checked.
Style	CSS Style sheet	Specifies an inline style for an element.
Subject	User define id	Specifies the element's corresponding item.
Tabindex	Tab number	Specifies the tab order of an element.
Title	User Defined	"Pop-up" title for your elements.
Valign	top, middle, bottom	Vertically aligns tags within an HTML element.
Width	Numeric Value	Specifies the width of tables, images, or table cells.

For a complete list of HTML5 Tags and related attributes, please check our reference to HTML5 Tags.

Custom Attributes

A new feature being introduced in HTML 5 is the addition of custom data attributes.

A custom data attribute starts with data- and would be named based on your requirement. Here is a simple example −

```
<div class = "example" data-subject = "physics" data-level = "complex">
   ...
</div>
```

The above code will be perfectly valid HTML5 with two custom attributes called datasubject and data-level. You would be able to get the values of these attributes using JavaScript APIs or CSS in similar way as you get for standard attributes.

HTML5 Events

When users visit your website, they perform various activities such as clicking on text and images and links, hover over defined elements, etc. These are examples of what JavaScript calls events.

We can write our event handlers in Javascript or VBscript and you can specify these event handlers as a value of event tag attribute. The HTML5 specification defines various event attributes as listed below − We can use the following set of attributes to trigger any javascript or vbscript code given as value when there is any event that takes place for any HTML5 element.

3.4 HTML5 Web Forms 2.0

Web Forms 2.0 is an extension to the forms features found in HTML4. Form elements and attributes in HTML5 provide a greater degree of semantic mark-up than HTML4 and free us from a great deal of tedious scripting and styling that was required in HTML4.
The <input> element in HTML4

HTML4 input elements use the type attribute to specify the data type.HTML4 provides following types –

No.	Type & Description
1	text A free-form text field, nominally free of line breaks.
2	password A free-form text field for sensitive information, nominally free of line breaks.
3	checkbox A set of zero or more values from a predefined list.
4	radio An enumerated value.
5	submit A free form of button initiates form submission.
6	file An arbitrary file with a MIME type and optionally a file name.
7	image A coordinate, relative to a particular image's size, with the extra semantic that it must be the last value selected and initiates form submission.
8	hidden An arbitrary string that is not normally displayed to the user.

9	select An enumerated value, much like the radio type.
10	textarea A free-form text field, nominally with no line break restrictions.
11	button A free form of button which can initiates any event related to button.

Following is the simple example of using labels, radio buttons, and submit buttons.

```
<form action = "http://example.com/cgiscript.pl" method = "post">
<p>
<label for = "firstname">first name: </label>
<input type = "text" id = "firstname"><br />

<label for = "lastname">last name: </label>
<input type = "text" id = "lastname"><br />

<label for = "email">email: </label>
<input type = "text" id = "email"><br>

<input type = "radio" name = "sex" value = "male"> Male<br>
<input type = "radio" name = "sex" value = "female"> Female<br>
<input type = "submit" value = "send"><input type = "reset">
</p>
</form>
```

...

The <input> element in HTML5

Apart from the above-mentioned attributes, HTML5 input elements introduced several new values for the type attribute. These are listed below.

NOTE − Try all the following example using latest version of Opera browser.

No.	Type & Description
1	datetime A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
2	datetime-local A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no

	time zone information.
3	date A date (year, month, day) encoded according to ISO 8601.
4	month A date consisting of a year and a month encoded according to ISO 8601.
5	week A date consisting of a year and a week number encoded according to ISO 8601.
6	time A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
7	number It accepts only numerical value. The step attribute specifies the precision, defaulting to 1.
8	range The range type is used for input fields that should contain a value from a range of numbers.
9	email It accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.
10	url It accepts only URL value. This type is used for input fields that should contain a URL address. If you try to submit a simple text, it forces to enter only URL address either in http://www.example.com format or in http://example.com format.

The <output> element

HTML5 introduced a new element <output> which is used to represent the result of different types of output, such as output written by a script.

You can use the for attribute to specify a relationship between the output element and other elements in the document that affected the calculation (for example, as inputs or parameters). The value of the for attribute is a space-separated list of IDs of other elements.

<!DOCTYPE HTML>

```
<html>
<head>
<script type = "text/javascript">

        function showResult() {
          x = document.forms["myform"]["newinput"].value;
document.forms["myform"]["result"].value = x;
        }
</script>
</head>

<body>

<form action = "/cgi-bin/html5.cgi" method = "get" name = "myform">
        Enter a value :<input type = "text" name = "newinput" />
<input type = "button" value = "Result"  onclick = "showResult();" />
<output name = "result"></output>
</form>

</body>
</html>
```

3.5 History of XHTML

EXtensible HyperText Markup Language (XHTML) is a mix of HTML and XML, very similar to HTML but stricter. It's like a rulebook for creating web pages that browsers easily understand. Unlike HTML, you have to be careful and follow the rules exactly. Most browsers support it. Just think of it as a more precise way to write web code.

It was developed by the World Wide Web Consortium (W3C) and helps web developers transition from HTML to XML. With XHTML, developers can enter the XML world with all its features while still ensuring backward and future compatibility of the content. The XHTML family includes three document types; the first is XHTML 1.0, which was recommended by W3C on January 26, 2000. The second is XHTML 1.1, which was recommended by W3C on May 31, 2001.

The third is XHTML5, a standard used for developing an XML adaptation of the HTML5 specification. An XHTML document must have an XHTML <!DOCTYPE> declaration.
Elements of XHTML:

XHTML Element	Description
<!DOCTYPE>	Used to declare the Document Type Definition (DTD), specifying the rules for the markup language, ensuring proper rendering in browsers.
<html>	Encloses the entire HTML or XHTML document, serving as the root element.
<head>	Contains meta-information about the document, such as the title, character set, linked stylesheets, and other essential elements.
<title>	Nested within the head section, specifies the title of the document, displayed in the browser's title bar or tab.
<body>	Encloses the content of the web page, including text, images, links, and other HTML elements. It represents the visible part of the document displayed in the browser.

When creating an XHTML web page, it is necessary to include a DTD (Document Type Definition) declaration. There are three types of DTD which are discussed below:

3.6 XHTLM Transitional DTD:

It is supported by the older browsers which do not have inbuilt cascading style sheets supports. Several attributes are enclosed in the body tag which are not allowed in strict DTD.
Syntax:
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

Example: In this example we will see the code for writing an XHTML document with an example.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
	<title>Transitional DTD XHTML</title>
</head>

<body bgcolor="#dae1ed">
	<div style="color:#090;font-size:40px;
				font-weight:bold;text-align:center;
				margin-bottom:-25px;">IFT 203 Course Materials</div>
	<p style="text-align:center;font-size:20px;">
		A computer science portal</p>
	<p style="text-align:center;font-size:20px;">
		Option to choose month:
		<select name="month">
			<option selected="selected">January</option>
			<option>February</option>
			<option>March</option>
			<option>April</option>
			<option>May</option>
			<option>June</option>
			<option>July</option>
			<option>Augusy</option>
			<option>September</option>
			<option>October</option>
			<option>November</option>
			<option>December</option>
		</select>
	</p>
</body>
</html>
```

Strict DTD:

Strict DTD is used when XHTML page contains only markup language. Strict DTD is used together with cascading style sheets, because this attribute does not allow CSS property in body tag.

Syntax:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
```

```
lang="en">
```

Example 2: In this example we will see the code for writing an XHTML document with an example for strict DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">

<head>
    <title>Strict DTD XHTML</title>
</head>

<body>
    <div style="color:#090;font-size:40px;
                font-weight:bold;text-align:center;
                margin-bottom:-25px;">IFT 203 Course
Materials</div>
    <p style="text-align:center;font-size:20px;">
        A computer science portal</p>
    <p style="text-align:center;font-size:20px;">
        Option to choose month:
        <select name="month">
            <option selected="selected">January</option>
            <option>February</option>
            <option>March</option>
            <option>April</option>
            <option>May</option>
            <option>June</option>
            <option>July</option>
            <option>Augusy</option>
            <option>September</option>
            <option>October</option>
            <option>November</option>
            <option>December</option>
        </select>
    </p>
</body>
</html>
```

3.7 XHTML Frameset DTD:

The frameset DTD is used when XHTML page contains frames. This DTD is identical to the HTML 4.01 Transitional DTD except for the content model of the HTML element.

Syntax:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en" lang="en">

<head>
    <title>Frameset DTD XHTML</title>
</head>
<frameset cols="30%, 20%, *">
    <frameset rows="40%, 30%, *">
        <frame src="gfg.html" />
        <frame src="gfg1.html" />
        <frame src="geeks.html" />
    </frameset>
    <frameset rows="40%, 60%">
        <frame src="g4g.html" />
        <frame src="g4g1.html" />
    </frameset>
    <frameset rows="20%, 20%, 30%, *">
        <frame src="IFT 203 Course Materials.html" />
        <frame src="IFT 203 Course Materials1.html" />
        <frame src="IFT 203 Course Materials2.html" />
        <frame src="IFT 203 Course Materials3.html" />
    </frameset>
</frameset>
</html>
```

Why use XHTML?

XHTML documents are validated with standard XML tools.
It is easy to maintain, convert, and edit documents in the long run.
It is used to define the quality standard of web pages.
XHTML is an official standard of the W3C, your website becomes more compatible and accurate with many browsers.

Benefits of XHTML:

All XHTML tags must have closing tags and are nested correctly. This generates cleaner code.

XHTML documents are lean which means they use less bandwidth. This reduces cost particularly if your web site has 1000s of pages.
XHTML documents are well formatted well–formed and can easily be transported to wireless devices, Braille readers and other specialized web environments.
All new developments will be in XML (of which XHTML is an application).
XHTML works in association with CSS to create web pages that can easily be updated.

Difference Between HTML and XHTML

HTML	**XHTML**
HTML or HyperText Markup Language is the main markup language for creating web pages	XHTML (Extensible HyperText Markup Language) is a family of XML markup languages that mirror or extend versions of the widely used Hypertext Markup Language (HTML)
Flexible framework requiring lenient HTML specific parser	Restrictive subset of XML which needs to be parsed with standard XML parsers
Proposed by Tim Berners-Lee in 1987	World Wide Web Consortium Recommendation in 2000.
Application of Standard Generalized Markup Language (SGML).	Application of XML
Extended from SGML.	Extended from XML, HTML

Self-Assessment Exercise(s)

(1) What does HTML stand for?
A. Hyperlinks and Text Markup Language
B. Home Tool Markup Language
C. Hyper Text Markup Language
D. Hyperlinks and Text Modeling Language
Answer: C. Hyper Text Markup Language

(2) Which of the following is true about XHTML?
A. XHTML is case-insensitive.
B. XHTML elements must be properly nested.
C. XHTML does not require closing tags.
D. XHTML is more lenient with coding errors compared to HTML.
Answer: B. XHTML elements must be properly nested.

(3) Which HTML tag is used to define an internal style sheet?
A. <style>
B. <css>
C. <script>
D. <link>
Answer: A. <style>

(4) In XHTML, which attribute must be present in every <img> tag to ensure the document is valid?
A. src
B. alt
C. title
D. width
Answer: B. alt

(5) Which of the following is a correct way to write a self-closing tag in XHTML?
A. <img src="image.jpg">
B.
C. <img src="image.jpg"></img>
D. <img src="image.jpg" /
Answer: B.

Conclusion

HTML5 and XHTML represent significant advancements in web development, each with distinct benefits and applications. HTML5, with its enhanced multimedia support, semantic elements, and backward compatibility, has become the preferred standard for modern web development due to its flexibility and comprehensive feature set. In contrast, XHTML provides a more stringent framework by enforcing strict XML-based syntax rules, ensuring well-formed and consistently rendered web documents. However, its rigidity can be challenging compared to the leniency of HTML5. Overall, while XHTML emphasizes structural accuracy and consistency, HTML5's balance of robustness and ease of use has made it the dominant choice for contemporary web design and development.

4.0 Summary

HTML5 is the most recent iteration of the HyperText Markup Language, which serves as the foundation for structuring and presenting content on the web. It was developed to improve and simplify multimedia handling, supporting audio, video, and interactive content directly within the browser without needing external plugins. HTML5 introduces semantic elements like <article>, <section>, and <nav>, enhancing the clarity and structure of web content. These additions not only improve code readability but also assist in better search engine optimization and accessibility. HTML5 is designed to be backward compatible, ensuring older content remains functional while providing modern web developers with new, powerful tools and features. XHTML blends the elements of HTML with the stricter syntax rules of XML, aiming to create more rigorously structured web documents. This standard ensures that web pages are well-formed, meaning that all tags must be properly nested and closed, and attributes must be correctly quoted. Such strictness facilitates consistent rendering across different browsers and platforms, reducing discrepancies. However, XHTML's rigidity can lead to challenges if the syntax is not precisely followed, making it less forgiving than HTML. With the advent of HTML5, which offers both leniency in syntax and a broad range of new features, XHTML has seen a decline in popularity despite its benefits in enforcing clean and reliable code.

5.0 References/Further Reading

Robbins, J. N. (2012). Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. " O'Reilly Media, Inc.".

McGrath, M. (2020). HTML in easy steps. In Easy Steps.

Tabarés, R. (2021). HTML5 and the evolution of HTML; tracing the origins of digital platforms. Technology in Society, 65, 101529.

Macaulay, M. (2017). Introduction to web interaction design: With Html and Css. Chapman and Hall/CRC.

Rebah, H. B., Boukthir, H., & Chedebois, A. (2022). Website Design and Development with HTML5 and CSS3. John Wiley & Sons.

MODULE 3 CASCADING STYLE SHEET (CSS) BASICS

MODULE INTRODUCTION

Cascading Style Sheets (CSS) is a cornerstone technology in web development, enabling the separation of content from presentation. By controlling the visual and aural layout of web pages, CSS enhances user experience and accessibility, allowing developers to create visually appealing and responsive designs. Understanding CSS is crucial for anyone looking to build or maintain modern websites, as it provides the tools to manipulate fonts, colors, layouts, and overall aesthetics with precision and efficiency. The CSS Basics module serves as an introductory guide to the fundamental concepts and techniques of CSS. This module covers essential topics such as selectors, properties, values, and the box model, providing a solid foundation for more advanced styling practices. Through practical examples and hands-on exercises, learners will gain the skills needed to apply CSS effectively, transforming plain HTML into dynamic, engaging web pages that work seamlessly across different devices and screen sizes. Whether you're a novice developer or looking to refresh your knowledge, this module will equip you with the core competencies required to harness the power of CSS in your web projects.

Unit 1 Introduction to Cascading Style Sheet (CSS)

Contents

1.0 Introduction

Cascading Style Sheets (CSS) are a cornerstone technology in modern web development, working in tandem with HTML and JavaScript to create visually appealing and interactive web pages. CSS is a style sheet language used for describing the presentation of a document written in HTML or XML. It controls the layout of multiple web pages all at once, allowing developers to create a consistent look and feel across a site with less code duplication. By separating content from design, CSS enhances flexibility and makes it easier to maintain websites. This unit will delve into the essentials of CSS, from its basic syntax and structure to advanced features such as responsive design.
In this unit, we will explore how CSS enhances web accessibility and performance, ensuring websites are not only aesthetically pleasing but also user-friendly and efficient. You will learn how to apply various styles to elements, manage layouts using the box model, and implement complex designs with techniques like Flexbox and Grid. Additionally, this unit will cover best practices for writing clean and maintainable CSS code, leveraging tools like preprocessors and frameworks. Whether you are a novice developer or looking to refine your skills, this unit aims to provide a comprehensive understanding of CSS and its critical role in modern web development.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss the meaning of CSS Syntax
- explain CSS Syntax
- identify simple CSS Elements
- use simple CSS elements like list, colour, and table etc in practical designs

3.0 Main Content

3.1 Cascading Style Sheets Introduction

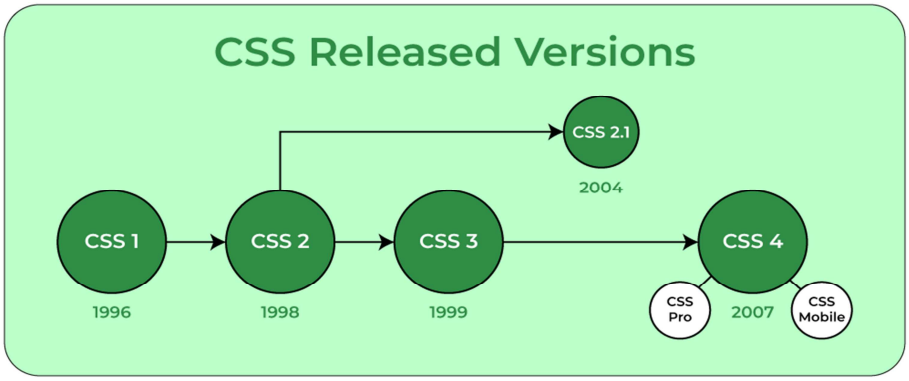
Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML, or XHTML). CSS describes how elements should be rendered on screen, paper, speech, or other media. CSS is a designed language intended to simplify the process of

making web pages presentable. CSS allows you to apply styles to HTML documents. It describes how a webpage should look. It prescribes colors, fonts, spacing, etc. In short, you can make your website look however you want. CSS lets developers and designers define how it behaves, including how elements are positioned in the browser.

HTML uses tags and CSS uses rulesets. CSS styles are applied to the HTML element using selectors. CSS is easy to learn and understand, but it provides powerful control over the presentation of an HTML document.

CSS is among the core languages of the **open web** and is standardized across Web browsers according to W3C specifications. Previously, the development of various parts of CSS specification was done synchronously, which allowed the versioning of the latest recommendations. You might have heard about CSS1, CSS2.1, or even CSS3. There will never be a CSS3 or a CSS4; everything is now CSS without a version number.

After CSS 2.1, the scope of the specification increased significantly and the progress on different CSS modules started to differ so much, that it became more effective to develop and release recommendations separately per module. Instead of versioning the CSS specification, W3C now periodically takes a snapshot of the latest stable state of the CSS specification and individual modules' progress. CSS modules now have version numbers, or levels, such as CSS Color Module Level 5.



Why CSS?
CSS saves time: You can write CSS once and reuse the same sheet in multiple HTML pages.

Easy Maintenance: To make a global change simply change the style, and all elements in all the webpages will be updated automatically.

Search Engines: CSS is considered a clean coding technique, which means search engines won't have to struggle to "read" its content.

Superior styles to HTML: CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.

Offline Browsing: CSS can store web applications locally with the help of an offline cache. Using this we can view offline websites.

3.2 CSS basics

CSS is the code that styles web content. *CSS basics* walks through what you need to get started. We'll answer questions like: How do I make text red? How do I make content display at a certain location in the (webpage) layout? How do I decorate my webpage with background images and colors? Like HTML, CSS is not a programming language. It's not a markup language either. CSS is what you use to selectively style HTML elements. For example, this CSS selects paragraph text, setting the color to red:

```
p {
  color: red;
}
```

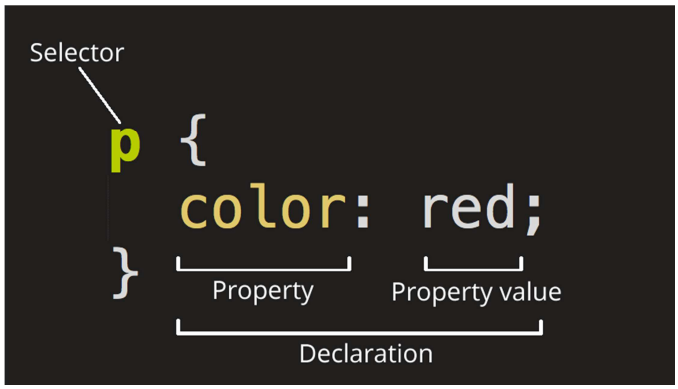
To make the code work, we still need to apply this CSS (above) to your HTML document. Otherwise, the styling won't change the appearance of the HTML.

Open your index.html file. Paste the following line in the head (between the <head> and </head> tags):

HTML

```
<link href="styles/style.css" rel="stylesheet" />
```

Let's dissect the CSS code for red paragraph text to understand how it works:



The whole structure is called a **ruleset**. (The term *ruleset* is often referred to as just *rule*.) Note the names of the individual parts:

Selector

This is the HTML element name at the start of the ruleset. It defines the element(s) to be styled (in this example, <p> elements). To style a different element, change the selector.

Declaration

This is a single rule like color: red;. It specifies which of the element's **properties** you want to style.

Properties

These are ways in which you can style an HTML element. (In this example, color is a property of the <p> elements.) In CSS, you choose which properties you want to affect in the rule.

Property value

To the right of the property—after the colon—there is the **property value**. This chooses one out of many possible appearances for a given property. (For example, there are many color values in addition to red.)

Note the other important parts of the syntax:

Apart from the selector, each ruleset must be wrapped in curly braces. ({})

Within each declaration, you must use a colon (:) to separate the property from its value or values.

Within each ruleset, you must use a semicolon (;) to separate each declaration from the next one.

To modify multiple property values in one ruleset, write them separated by semicolons, like this:

CSSCopy to Clipboard

```
p {
  color: red;
  width: 500px;
  border: 1px solid black;
}
```

Selecting multiple elements

You can also select multiple elements and apply a single ruleset to all of them. Separate multiple selectors by commas. For example:

CSSCopy to Clipboard

```
p,
li,
h1 {
  color: red;
}
```

Different types of selectors

There are many different types of selectors. The examples above use **element selectors**, which select all elements of a given type. But we can make more specific selections as well. Here are some of the more common types of selectors:

Selector name	What does it select	Example
Element selector (sometimes called a	All HTML elements of the specified type.	p selects <p>

Selector name	What does it select	Example
tag or type selector)		
ID selector	The element on the page with the specified ID. On a given HTML page, each id value should be unique.	#my-id selects <p id="my-id"> or <a id="my-id">
Class selector	The element(s) on the page with the specified class. Multiple instances of the same class can appear on a page.	.my-class selects <p class="my-class"> and <a class="my-class">
Attribute selector	The element(s) on the page with the specified attribute.	img[src] selects <img src="myimage.png"> but not <img>
Pseudo-class selector	The specified element(s), but only when in the specified state. (For example, when a cursor hovers over a link.)	a:hover selects <a>, but only when the mouse pointer is hovering over the link.

Fonts and text

Now that we've explored some CSS fundamentals, let's improve the appearance of the example by adding more rules and information to the style.css file.

First, find the output from Google Fonts that you previously saved. What will your website look like? Add the <link> element somewhere inside your index.html's head (anywhere between the <head> and </head> tags). It looks something like this:

```
<link
  href="https://fonts.googleapis.com/css?family=Open+Sans"
  rel="stylesheet" />
```

This code links your page to a style sheet that loads the Open Sans font family with your webpage.

Next, delete the existing rule you have in your style.css file. It was a good test, but let's not continue with lots of red text.

Add the following lines (shown below), replacing the font-family assignment with your font-family selection from What will your website look like? The property font family refers to the font(s) you want to use for text. This rule defines a global base font and font size

for the whole page. Since <html> is the parent element of the whole page, all elements inside it inherit the same font size and font family.

```
HTML {
  font-size 10px; /* px means "pixels": the base font size is now 10 pixels high */
  font-family: "Open Sans", sans-serif; /* this should be the rest of the output you got from Google Fonts */
}
```

Note: Anything in CSS between /* and */ is a **CSS comment**. The browser ignores comments as it renders the code. CSS comments are a way for you to write helpful notes about your code or logic.

Now let's set font sizes for elements that will have text inside the HTML body (<h1>, <li>, and <p>). We'll also center the heading. Finally, let's expand the second ruleset (below) with settings for line height and letter spacing to make body content more readable.

```
h1 {
  font-size: 60px;
  text-align: center;
}

p,
li {
  font-size: 16px;
  line-height: 2;
  letter-spacing: 1px;
}
```

Adjust the px values as you like. Your work-in-progress should look similar to this:

CSS: all about boxes

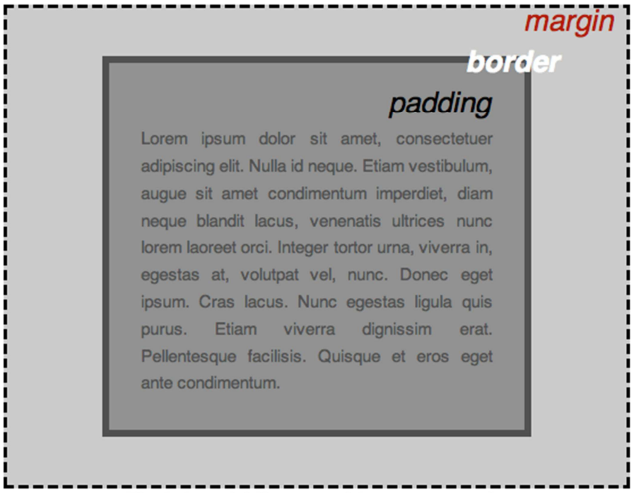
Something you'll notice about writing CSS: a lot of it is about boxes. This includes setting size, color, and position. Most HTML elements on your page can be thought of as boxes sitting on top of other boxes.

CSS layout is mostly based on the *box model*. Each box taking up space on your page has properties like:

padding, the space around the content. In the example below, it is the space around the paragraph text.

border, the solid line that is just outside the padding.

margin, the space around the outside of the border.



In this section we also use:

width (of an element).

background-color, the color behind an element's content and padding.

color, the color of an element's content (usually text).

text-shadow sets a drop shadow on the text inside an element.

display sets the display mode of an element. (keep reading to learn more)

To continue, let's add more CSS. Keep adding these new rules at the bottom of style.css. Experiment with changing values to see what happens.

Changing the page color

```
html {
  background-color: #00539f;
}
```

This rule sets a background color for the entire page. Change the color code to the color you chose in What will my website look like?

Styling the body

```
body {
  width: 600px;
  margin: 0 auto;
  background-color: #ff9500;
  padding: 0 20px 20px 20px;
  border: 5px solid black;
}
```

There are several declarations for the <body> element. Let's go through this line-by-line:

width: 600px; This forces the body to always be 600 pixels wide.

margin: 0 auto; When you set two values on a property like margin or padding, the first value affects the element's top *and* bottom side (setting it to 0 in this case); the second value affects the left *and* right side. (Here, auto is a special value that divides the available horizontal space evenly between left and right). You can also use one, two, three, or four values, as documented in Margin Syntax.

background color: #FF9500; This sets the element's background color. This project uses a reddish orange for the body background color, as opposed to dark blue for the <html> element. (Feel free to experiment.)

padding: 0 20px 20px 20px; This sets four values for padding. The goal is to put some space around the content. In this example, there is no padding on the top of the body, and 20 pixels on the right, bottom and left. The values set top, right, bottom, left, in that order. As with margin, you can use one, two, three, or four values, as documented in Padding Syntax.

border: 5px solid black; This sets values for the width, style and color of the border. In this case, it's a five-pixel–wide, solid black border, on all sides of the body.

Positioning and styling the main page title

```
h1 {
  margin: 0;
  padding: 20px 0;
  color: #00539f;
```

```
  text-shadow: 3px 3px 1px black;
}
```

You may have noticed there's a horrible gap at the top of the body. That happens because browsers apply default styling to the h1 element (among others). That might seem like a bad idea, but the intent is to provide basic readability for unstyled pages. To eliminate the gap, we overwrite the browser's default styling with the setting margin: 0;.

Next, we set the heading's top and bottom padding to 20 pixels.

Following that, we set the heading text to be the same color as the HTML background color.

Finally, text-shadow applies a shadow to the text content of the element. Its four values are:

The first pixel value sets the **horizontal offset** of the shadow from the text: how far it moves across.

The second pixel value sets the **vertical offset** of the shadow from the text: how far it moves down.

The third pixel value sets the **blur radius** of the shadow. A larger value produces a more fuzzy-looking shadow.

The fourth value sets the base color of the shadow.

Try experimenting with different values to see how it changes the appearance.

Centering the image

```
img {
  display: block;
  margin: 0 auto;
}
```

Next, we center the image to make it look better. We could use the margin: 0 auto trick again as we did for the body. But there are differences that require an additional setting to make the CSS work.

The <body> is a **block** element, meaning it takes up space on the page. The margin applied to a block element will be respected by other elements on the page. In contrast, images are **inline** elements, for the auto margin trick to work on this image, we must give it block-level behavior using display: block;.

Note: The instructions above assume that you're using an image smaller than the width set on the body. (600 pixels) If your image is larger, it will overflow the body, spilling into the rest of the page. To fix this, you can either: 1) reduce the image width using a graphics editor, or 2) use CSS to size the image by setting the width property on the <img> element with a smaller value.

Note: Don't be too concerned if you don't completely understand display: block; or the differences between a block element and an inline element. It will make more sense as you continue your study of CSS.

Syntax

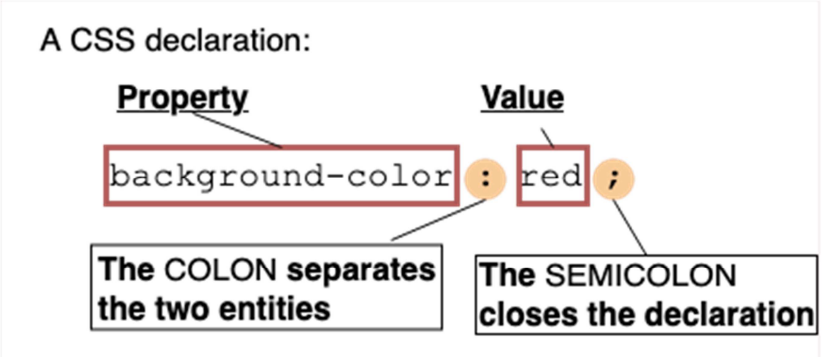
The basic goal of the CSS language is to allow a browser engine to paint elements of the page with specific features, like colors, positioning, or decorations. The *CSS syntax* reflects this goal and its basic building blocks are:

The **property** which is an identifier, that is a human-readable *name*, that defines which feature is considered.

The **value** which describe how the feature must be handled by the engine. Each property has a set of valid values, defined by a formal grammar, as well as a semantic meaning, implemented by the browser engine.

CSS declarations

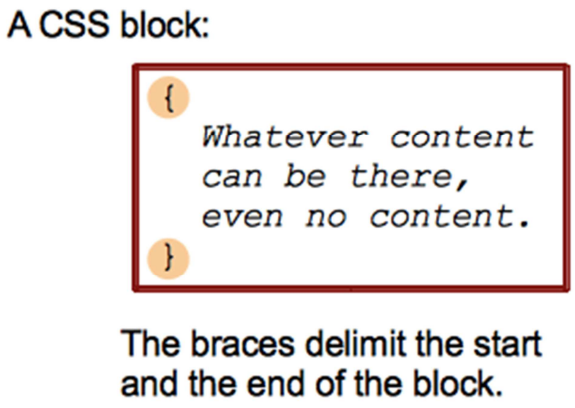
Setting CSS properties to specific values is the core function of the CSS language. A property and value pair is called a **declaration**, and any CSS engine calculates which declarations apply to every single element of a page in order to appropriately lay it out, and to style it.
Both properties and values are case-insensitive by default in CSS. The pair is separated by a colon, ':' (U+003A COLON), and white spaces before, between, and after properties and values, but not necessarily inside, are ignored.



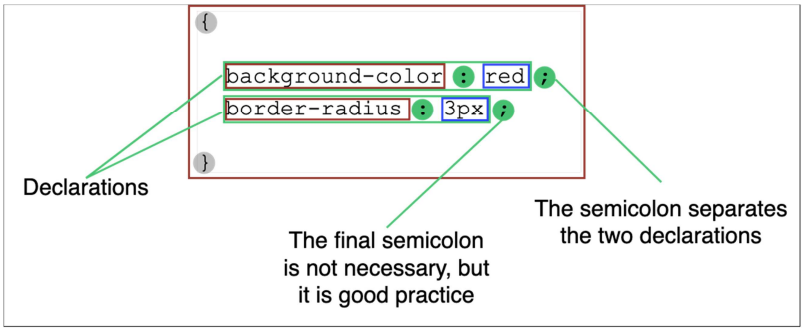
There are hundreds of different properties in CSS and a practically endless number of different values. Not all pairs of properties and values are allowed and each property defines what are the valid values. When a value is not valid for a given property, the declaration is deemed *invalid* and is wholly ignored by the CSS engine.

CSS declaration blocks

Declarations are grouped in **blocks**, that is in a structure delimited by an opening brace, '{' (U+007B LEFT CURLY BRACKET), and a closing one, '}' (U+007D RIGHT CURLY BRACKET). Blocks sometimes can be nested, so opening and closing braces must be matched.



Such blocks are naturally called **declaration blocks** and declarations inside them are separated by a semicolon, ';' (U+003B SEMICOLON). A declaration block may be empty, that is containing null declaration. White spaces around declarations are ignored. The last declaration of a block doesn't need to be terminated by a semicolon, though it is often considered *good style* to do it as it prevents forgetting to add it when extending the block with another declaration.
A CSS declaration block is visualized in the diagram below.

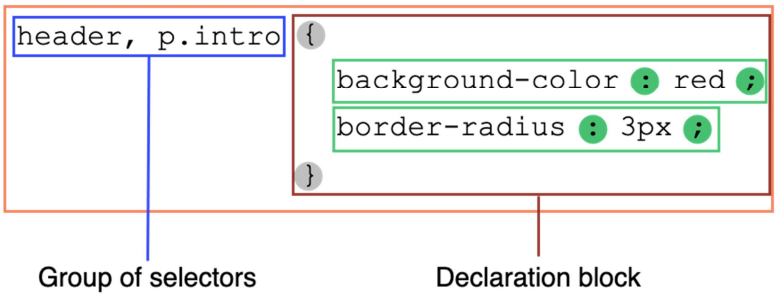


Note: The content of a CSS declaration block, that is a list of semicolon-separated declarations, without the initial and closing braces, can be put inside an HTML style attribute.

CSS rulesets
If style sheets could only apply a declaration to each element of a Web page, they would be pretty useless. The real goal is to apply different declarations to different parts of the document.

CSS allows this by associating conditions with declarations blocks. Each (valid) declaration block is preceded by one or more comma-separated **selectors**, which are conditions selecting some elements of the page. A selector list and an associated declarations block, together, are called a **ruleset**, or often a **rule**.

A CSS ruleset (or rule) is visualized in the diagram below.



As an element of the page may be matched by several selectors, and therefore by several rules potentially containing a given property several times, with different values, the CSS standard defines which one has precedence over the other and must be applied: this is called the cascade algorithm.

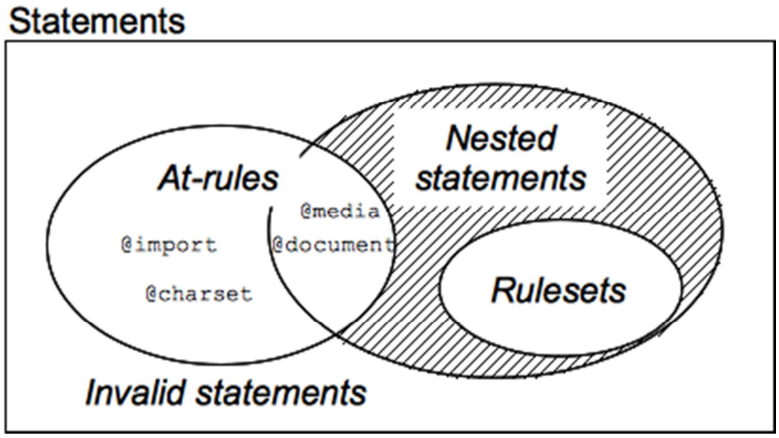
Note: It is important to note that even if a ruleset characterized by a group of selectors is a kind of shorthand replacing rulesets with a single selector each, this doesn't apply to the validity of the ruleset itself.

This leads to an important consequence: if one single basic selector is invalid, like when using an unknown pseudo-element or pseudo-class, the whole *selector* is invalid and therefore the entire rule is ignored (as invalid too).

3.3 CSS statements

Rulesets are the main building blocks of a style sheet, which often consists of only a big list of them. But there is other information that a Web author wants to convey in the style sheet, like the character set, other external style sheets to import, font face or list counter descriptions and many more. It will use other and specific kinds of statements to do that.

A **statement** is a building block that begins with any non-space characters and ends at the first closing brace or semicolon (outside a string, non-escaped, and not included into another {}, () or [] pair).



There are two kinds of statements:
Rulesets (or *rules*) that, as seen, associate a collection of CSS declarations to a condition described by a selector.

At-rules that start with an at sign, '@' (U+0040 COMMERCIAL AT), followed by an identifier and then continuing up to the end of the statement, that is up to the next semicolon (;) outside of a block, or the end of the next block. Each type of at-rules, defined by the identifier, may have its own internal syntax, and semantics of course. They are used to convey meta-data information (like @charset or @import), conditional information (like @media or @document), or descriptive information (like @font-face).

Any statement that isn't a ruleset or an at-rule is invalid and ignored.

Nested statements
There is another group of statements – the **nested statements**. These are statements that can be used in a specific subset of at-rules – the *conditional group rules*. These statements only apply if a specific condition is matched: the @media at-rule content is applied only if the device on which the browser runs matches the expressed condition; the @document at-rule content is applied only if the current page matches some conditions, and so on. In CSS1 and CSS2.1, only *rulesets* could be used inside conditional group rules. That was very restrictive and this restriction was lifted in *CSS Conditionals Level 3*. Now, though still experimental and not supported by every browser, conditional group rules can contain a wider range of content: rulesets but also some, but not all, at-rules.

CSS comprises style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule set consists of a selector and declaration block.

Selector: A selector in CSS is used to target and select specific HTML elements to apply styles.

Declaration: A declaration in CSS is a combination of a property and its corresponding value.

```
// HTML Element
<h1>IFT 203 Class</h2>

// CSS Style
h1 { color: blue; font-size: 12px; }
```

Where -
Selector - h1
Declaration - { color: blue; font-size: 12px; }
The selector points to the HTML element that you want to style.
The declaration block contains one or more declarations separated by semicolons.
Each declaration includes a CSS property name and a value, separated by a colon.
Example
CSS

```
p {
        color: blue;
        text-align: center;
}
```

CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces. In this example, all paragraph element (<p> tag) will be centre-aligned, with a blue text color.
Web Page with & without CSS
Without CSS: In this example, we have not added any CSS style.
html

```
<!DOCTYPE html>
<html>

<head>
        <title>Simple Web Page</title>
</head>

<body>
        <main>
        <h1>HTML Page</h1>
        <p>This is a basic web page.</p>
        </main>
</body>
</html>
```

Using CSS: In this example, we will add some CSS styles inside the HTML document to show how CSS makes a HTML page attractive and user-friendly.

html

```
<!DOCTYPE html>
<html>

<head>
	<title>Simple web page</title>
	<style>
		main {
			width: 600px;
			height: 200px;
			padding: 10px;
			background: beige;
		}

		h1 {
			color: olivedrab;
			border-bottom: 1px dotted darkgreen;
		}

		p {
			font-family: sans-serif;
			color: orange;
		}
	</style>
</head>

<body>
	<main>
		<h1>My first Page</h1>
		<p>This is a basic web page.</p>
	</main>
</body>
</html>
```

3.4 CSS Comments

CSS comments are specific lines within your code that the compiler intentionally ignores, ensuring they are not executed. The fundamental objective of incorporating comments is to enhance the readability and comprehensibility of your code, making it more user-friendly for fellow developers. This practice is not only beneficial for team collaborations but also important for individual coders who revisit their

code after a certain period. Remember, well-commented code is a hallmark of professional web development.

Syntax:

/* content */

Comments can be single-line or multi-line. The /* */ comment syntax can be used for both single and multiline comments. We may use <!– –> syntax for hiding in CSS for older browsers, but this is no longer recommended for use.

Adding comments to the code is a good practice that can help to understand the code if someone reads the code or if it is reviewed later.

Note: The outputs are the same because comments are ignored by the browser and are not interpreted.

Examples of CSS Comments

Example 1: This example describes the single-line comment.

```
<!DOCTYPE html>
<html>
<head>
<style>
      h1 {
         color: green;
      }
      /* Single line comment */
</style>
</head>
<body>
<h1>IFT 203 Class</h1>
<p> A Computer Science portal for NOUN </p>
</body>
</html>
```

Example 2: This example describes the multi-line comment.

```
<!DOCTYPE html>
<html>
<head>
<style>
   h1 {
      color: green;
   }
   /* This is a multiline
         comment */
</style>
</head>
<body>
<h1>IFT 203 Class</h1>
<p> A Computer Science portal for NOUN </p>
</body>
</html>
```

3.5 CSS Colors

CSS Colors are an essential part of web design, providing the ability to bring your HTML elements to life. This feature allows developers to set the color of various HTML elements, including font color, background color, and more.

There are several ways to define the color of an element in CSS:
Built-In Color: These are a set of predefined colors which are used by their names. For example: red, blue, green etc.

RGB Format: The RGB (Red, Green, Blue) format is used to define the color of an HTML element by specifying the R, G, and B values range between 0 to 255.

RGBA Format: The RGBA format is similar to the RGB, but it includes an Alpha component that specifies the transparency of elements.

Hexadecimal Notation: The hexadecimal notation begins with a # symbol followed by 6 characters each ranging from 0 to F.

HSL: HSL stands for Hue, Saturation, and Lightness respectively. This format uses the cylindrical coordinate system.
HSLA: The HSLA color property is similar to the HSL property, but it includes an Alpha component that specifies the transparency of elements.
By understanding and utilizing these different methods, you can create vibrant and dynamic web pages that captivate your audience. Let's dive in and explore the CSS Colors!
Built-In Color: These are a set of predefined colors which are used by its name. For example: red, blue, green etc.
Syntax:

```
h1 {
   color: color-name;
}
```

Example:

```
<!DOCTYPE html>
<html>

<head>
<title>CSS color property</title>
<style>
      h1 {
         color: green;
         text-align: center;
```

```
    }
</style>
</head>

<body>
<h1>
    IFT 203 class
</h1>
</body>

</html>
```

RGB Format: The RGB(Red, Green, Blue) format is used to define the color of an HTML element by specifying the R, G, B values range between 0 to 255. For example: RGB value of Red color is (255, 0, 0), Green color is (0, 255, 0), Blue color is (0, 0, 255) etc.

Syntax:

```
h1 {
  color: rgb(R, G, B);
}
```

Example:

```
<!DOCTYPE html>
<html>

<head>
<title>CSS color property</title>
<style>
    h1 {
      color: rgb(0, 153, 0);
      text-align: center;
    }
</style>
</head>

<body>
<h1>
    IFT 2023 class
</h1>
</body>
</html>
```

RGBA Format: The RGBA format is similar to the RGB, but the difference is RGBA contains A (Alpha) which specifies the transparency of elements. The value of alpha lies between 0.0 to 1.0 where 0.0. represents fully transparent and 1.0 represents not transparent.

Syntax:

```
h1 {
```

```
   color:rgba(R, G, B, A);
}
```

Example:

```
<!DOCTYPE html>
<html>

<head>
<title>CSS RGBA color property</title>
<style>
      h1 {
         color: rgba(0, 153, 0, 0.5);
         text-align: center;
      }
</style>
</head>

<body>
<h1>
      IFT 203 Class
</h1>
</body>

</html>
```

Hexadecimal Notation: The hexadecimal notation begins with # symbol followed by 6 characters each ranging from 0 to F. For example: Red #FF0000, Green #00FF00, Blue #0000FF etc.

Syntax:

```
h1 {

   color:#(0-F)(0-F)(0-F)(0-F)(0-F)(0-F);
}
```

Example:

```
<!DOCTYPE html>
<html>

<head>
<title>CSS hex property</title>
<style>
      h1 {
         color: #009900;
         text-align: center;
      }
</style>
</head>
```

```
<body>
<h1>
      IFT 203 class
</h1>
</body>

</html>
```

HSL: HSL stands for Hue, Saturation, and Lightness respectively. This format uses the cylindrical coordinate system.
Hue: Hue is the degree of the color wheel. Its value lies between 0 to 360 where 0 represents red, 120 represents green and 240 represents blue color.
Saturation: It takes a percentage value, where 100% represents completely saturated, while 0% represents completely unsaturated (gray).
Lightness: It takes percentage value, where 100% represents white, while 0% represents black.
Syntax:

```
h1 {

  color:hsl(H, S, L);
}
```

Example:

```
<!DOCTYPE html>
<html>

<head>
<title>CSS hsl color property</title>
<style>
      h1 {
         color: hsl(120, 100%, 30%);
         text-align: center;
      }
</style>
</head>

<body>
<h1>
      IFT 203 Class
</h1>
</body>

</html>
```

HSLA:
The HSLA color property is similar to HSL property, but the difference is HSLA contains A (Alpha) which specifies the

transparency of elements. The value of alpha lies between 0.0 to 1.0 where 0.0. represents fully transparent and 1.0 represents not transparent.
Syntax:

```
h1 {
   color:hsla(H, S, L, A);
}
```

Example:

```
<!DOCTYPE html>
<html>

<head>
<title>CSS hsla color property</title>
<style>
      h1 {
         color: hsla(120, 100%, 50%, 0.50);
         text-align: center;
      }
</style>
</head>

<body>
<h1>
      IFT 203 class
</h1>
</body>

</html>
```

3.6 CSS Borders

CSS borders are essential elements in websites, representing the edges of various components and elements. CSS Borders refer to the lines that surround elements, defining their edges. Borders can be styled, colored, and sized using CSS properties such as border style, border color, border width, and border radius. borders can be styled with the top border, the right border, the bottom border, and the left border.

Common Border Styles
The border-style property specifies the type of border. None of the other border properties will work without setting the border style.
Following are the types of borders:
Dotted: Creates a series of dots.
Dashed: Forms a dashed line.
Solid: Produces a continuous line.
Double: Renders two parallel lines.

Groove and **Ridge**: Create 3D grooved and ridged effects.
Inset and **Outset**: Add 3D inset and outset borders.
None: Removes the border.
Hidden: Hides the border.
Examples of CSS border Style
In this example we are going to use CSS border-style property.

```
<!DOCTYPE html>
<html>

<head>
<style>
p.dotted {
        border-style: dotted;
    }

p.dashed {
        border-style: dashed;
    }

p.solid {
        border-style: solid;
    }

p.double {
        border-style: double;
    }
</style>
</head>

<body>
<h2>The border-style Property</h2>

<p>IFT 203 Class</p>

<p class="dotted">A dotted border.</p>

<p class="dashed">A dashed border.</p>

<p class="solid">A solid border.</p>

<p class="double">A double border.</p>

</body>

</html>
```

Explanation:

Here we defines paragraph elements with different border styles: dotted, dashed, solid, and double.
Each paragraph demonstrates a distinct border style applied through the border-style property.
CSS classes are used to assign specific border styles to paragraphs, such as .dotted, .dashed, .solid, and .double.
When rendered, each paragraph showcases its designated border style, enhancing visual presentation.
CSS Border Width
Border width sets the width of the border. The width of the border can be in px, pt, cm or thin, medium, and thick.
Example of Border Width
Here is the basic example of using CSS border width property.

```
<!DOCTYPE html>
<html>

<head>
<style>
      p {
         border-style: solid;
         border-width: 8px;
      }
</style>
</head>

<body>
<p>
      IFT 203 Class
</p>
<p>
      Border properties
</p>
</body>

</html>
```

Explanation:
In this example we contains two paragraphs styled with a solid border using CSS.
The border width for both paragraphs is set to 8 pixels, defined by the border-width property.
Each paragraph displays a solid border surrounding its content, enhancing visual separation.
The border width can be adjusted to modify the thickness of the border as desired.

CSS Border Color

This property is used to set the color of the border. Color can be set using the color name, hex value, or RGB value. If the color is not specified border inherits the color of the element itself.

Example of CSS Border Color

Here we are implementing above explained border color property.

```
<!DOCTYPE html>
<html>

<head>
<style>
      p {
         border-style: solid;
         border-color: red
      }
</style>
</head>

<body>
<p>
      IFT 203 Class
</p>
<p>
      Border properties:color
</p>
</body>

</html>
```

Explanation:

Here we includes two paragraphs styled with a solid border.

The border color for both paragraphs is set to red using the border-color property.

Each paragraph displays a solid red border around its content, enhancing visual distinction.

Border color can be customized by specifying different color values.

CSS Border individual sides:

Using border property, we can provide width, style, and color to all the borders separately for that we have to give some values to all sides of the border.

CSS Border individual sides Syntax:

```
border-top-style : dotted;
border-bottom-width: thick;
border-right-color: green;
```

CSS Border individual sides Example:

In this example, we set border-top-style as dotted in h2.

```
<!DOCTYPE html>
```

```
<html>

<head>
<style>
      h2 {
         border-top-style: dotted;
      }
</style>
</head>

<body>
<h2>Welcome to IFT 203 Class </h2>
</body>

</html>
```

Explanation:
In thw above example we contains a single <h2> heading styled with a dotted border on the top.
The border style for the top of the heading is set using the border-top-style property.
This results in the top border of the heading being displayed as a dotted line.
Using CSS, borders can be styled in various ways to enhance the appearance of elements.

Border radius property
The CSS border-radius property rounds the corners of an element's border, creating smoother edges, with values specifying the curvature radius.
Border radius property Syntax:
border-radius: value;
Example of Border radius property
Here is the basic example of using border-radius property.

```
<!DOCTYPE html>
<html>

<head>
<style>
      h1 {
         border-style: solid;
         text-align: center;
         background: green;
         border-radius: 20px;
      }
</style>
</head>
```

```
<body>
<h1> IFT 203 Class </h1>
</body>
</html>
```

Explanation:
In the above example we contains a single <h1> heading styled with a solid border.
The text inside the heading is centered horizontally using the text-align property.
The heading has a green background color and rounded corners achieved with border-radius.
This creates a visually appealing header element with a solid border and rounded corners.
CSS Links
A link is a connection from one web page to another web page. CSS property can be used to style the links in various different ways. States of Link: Before discussing CSS properties, it is important to know the states of a link. Links can exist in different states and they can be styled using pseudo-classes.
There are four states of links given below:
a:link => This is a normal, unvisited link.
a:visited => This is a link visited by user at least once
a:hover => This is a link when mouse hovers over it
a:active => This is a link that is just clicked.
Syntax:

```
a:link {
   color:color_name;
}
```

color_name can be given in any format like color name (green), HEX value (#5570f0) or RGB value rgb(25, 255, 2). There is another state 'a:focus' which is used to focused when a user uses the tab key to navigate through the links.
The default value of links:
By default the links created are underlined.
When the mouse is hovered above a link, it changes to a hand icon.
Normal/unvisited links are blue.
Visited links are colored purple.
Active links are colored red.
When a link is focused, it has an outline around it.
Example: This example shows the basic use of links in CSS.

```
<!DOCTYPE html>
<html>

<head>
<title>CSS links</title>
<style>
```

```
        p {
            font-size: 25px;
            text-align: center;
        }
</style>
</head>

<body>
<p>
<a href="https://www.noun.edu.ng/">
            IFT 203 Class Simple Link
</a>
</p>
</body>

</html>
```

CSS Properties of Links: Some basic CSS properties of links are given below:

color
font-family
text-decoration
background-color

color: This CSS property is used to change the color of the link text.

Syntax:

```
a {
   color: color_name;
}
```

Example: This example shows the use of the color property in links.

```
<!DOCTYPE html>
<html>

<head>
<title>Link color property</title>
<style>
        p {
            font-size: 20px;
            text-align: center;
        }

        /*unvisited link will appear green*/
a:link {
            color: red;
        }

        /*visited link will appear blue*/
a:visited {
```

```
        color: blue;
    }

    /*when mouse hovers over link it will appear orange*/
a:hover {
        color: orange;
    }

    /*when the link is clicked, it will appear black*/
a:active {
        color: black;
    }
</style>
</head>

<body>
<p>
<a href="https://www.noun.edu.ng/">
        IFT 203 Class
</a>
    This link will change colours with different states.
</p>
</body>

</html>
```

font-family: This property is used to change the font type of a link using font-family property.

Syntax:

```
a {
   font-family: "family name";
}
```

Example: This example shows the use of the font-family property in links.

```
<!DOCTYPE html>
<html>

<head>
<style>
    /*Initial link font family arial*/
    a {
        font-family: Arial;
    }

    p {
        font-size: 30px;
        text-align: center;
```

```
        }

        /*unvisited link font family*/
a:link {
            color: Arial;
        }

        /*visited link font family*/
a:visited {
            font-family: Arial;
        }

        /*when mouse hovers over it will change to times new roman*/
a:hover {
            font-family: Times new roman;
        }

        /*when the link is clicked, it will changed to Comic sans ms*/
a:active {
            font-family: Comic Sans MS;
        }
</style>
</head>

<body>
<p>
<a href="https://www.noun.edu.ng/" id="link">
            IFT 203 Class
</a>
        a Computer Science Portal for Geeks.
</p>
</body>

</html>
```

Text-Decoration: This property is basically used to remove/add underlines from/to a link.

Syntax:

```
a {
   text-decoration: none;
}
```

Example: This example shows the use of the text-decoration property in links.

Text-Decoration: This property is basically used to remove/add underlines from/to a link.

Syntax:

```
a {
   text-decoration: none;
}
```

Example: This example shows the use of the text-decoration property in links.

```
<!DOCTYPE html>
<html>

<head>
<title>Text decoration in link</title>
<style>
      /*Set the font size for better visibility*/
      p {
         font-size: 2rem;
      }

      /*Removing underline using text-decoration*/
      a {
         text-decoration: none;
      }

      /*underline can be added using
      text-decoration:underline;
      */
</style>
</head>

<body>
<p>
<a href="https://www.noun.edu.ng/" id="link">
         IFT 203 Class
</a>
      a Computer Science Portal for Geeks.
</p>
</body>

</html>
```

background-color: This property is used to set the background color of the link.

Syntax:

```
a {
   background-color: color_name;
}
```

Example: This example shows the use of the background-color property in links.

```
<!DOCTYPE html>
```

```
<html>

<head>
<title>background color</title>
<style>
      /*Setting font size for better visibility*/
      p {
         font-size: 2rem;
      }

      /*Designing unvisited link button*/
a:link {
         background-color: powderblue;
         color: green;
         padding: 5px 5px;
         text-decoration: none;
         display: inline-block;
      }

      /*Designing link button when mouse cursor hovers over it*/
a:hover {
         background-color: green;
         color: white;
         padding: 5px 5px;
         text-align: center;
         text-decoration: none;
         display: inline-block;
      }
</style>
</head>

<body>
<p>
<a href="https://www.IFT 203 Class.org/" id="link">
         IFT 203 Class
</a>
      a Computer Science Portal for Geeks.
</p>
</body>

</html>
```

CSS Link Button: CSS links can also be styled using buttons/boxes. The following example shows how CSS links can be designed as buttons.

Example: This example shows the use of links as a button.

```
<!DOCTYPE html>
```

```
<html>

<head>
<title>Link button</title>
<style>
      /*Setting font size for better visibility*/
      p {
         font-size: 2rem;
      }

      a {
         background-color: green;
         color: white;
         padding: 5px 5px;
         border-radius: 5px;
         text-align: center;
         text-decoration: none;
         display: inline-block;
      }
</style>
</head>

<body>
<p>
<a href="https://www.noun.edu.ng/" id="link">
         IFT 203 Class
</a>
      a Computer Science Portal for Geeks.
</p>
</body>

</html>
```

3.7 CSS Lists

The **List** in CSS specifies the listing of the contents or items in a particular manner i.e., it can either be organized orderly or unorder way, which helps to make a clean webpage. It can be used to arrange the huge with a variety of content as they are flexible and easy to manage. The default style for the list is borderless.

The list can be categorized into 2 types:
Unordered List: In unordered lists, the list items are marked with bullets i.e. small black circles by default.

Ordered List: In ordered lists, the list items are marked with numbers and an alphabet.

We have the following CSS lists properties, which can be used to control the CSS lists:
list-style-type: This property is used to specify the appearance (such as disc, character, or custom counter style) of the list item marker.
list-style-image: This property is used to set the images that will be used as the list item marker.

list-style-position: It specifies the position of the marker box with respect to the principal block box.

list-style: This property is used to set the list style.
Now, we will learn more about these properties with examples.

List Item Marker
This property specifies the type of item marker i.e. unordered list or ordered. The list-style-type property specifies the appearance of the list item marker (such as a disc, character, or custom counter style) of a list item element. Its default value is a disc.

Syntax:
list-style-type: value;
The following value can be used:

CSS List Style Type	Description
none	No marker or bullet is displayed.
circle	Default marker for unordered lists, displays a hollow circle.
decimal	Default marker for ordered lists, displays decimal numbers (1, 2, 3, etc.).
decimal-leading-zero	Displays decimal numbers with leading zeroes (01, 02, 03, etc.).
lower-roman	Displays lowercase Roman numerals (i, ii, iii, etc.) for ordered lists.
upper-roman	Displays uppercase Roman numerals (I, II, III, etc.) for ordered lists.

CSS List Style Type	Description
lower-alpha	Displays lowercase alphabetical characters (a, b, c, etc.) for ordered lists.
upper-alpha	Displays uppercase alphabetical characters (A, B, C, etc.) for ordered lists.
square	Displays filled square markers for unordered lists.

CSS Lists Examples

Example 1: This example describes the CSS List with the various list-style-type where the values are set to square & lower-alpha.

```
<!DOCTYPE html>
<html>

<head>
<style>
ul.a {
        list-style-type: square;
      }

      ol.c {
        list-style-type: lower-alpha;
      }
</style>
</head>

<body>
<h2>
      IFT 203 Class
</h2>
<p> Unordered lists </p>
<ul class="a">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
<ul class="b">
<li>one</li>
<li>two</li>
<li>three</li>
```

```
</ul>
<p> Ordered Lists </p>
<ol class="c">
<li>one</li>
<li>two</li>
<li>three</li>
</ol>
<ol class="d">
<li>one</li>
<li>two</li>
<li>three</li>
</ol>
</body>

</html>
```

Example 2: This example describes the CSS List with the various list-style-image where the values are set to url of the image.

```
<!DOCTYPE html>
<html>

<head>
<title> CSS list-style-image Property </title>
<style>
      ul {
         list-style-image:
url("https://contribute.noun.edu.ng/wp-content/uploads/listitem-
1.png");
      }
</style>
</head>

<body>
<h1>
      IFT 203 Class
</h1>
<p> Unordered lists </p>
<ul>
<li>1</li>
<li>2</li>
<li>3</li>
</ul>
</body>

</html>
```

Styling Lists: The list can be formatted in CSS. Different colors, borders, backgrounds, and paddings can be set for the lists.

Example 3: This example describes the CSS List where the various styling properties are applied to the element.

```
<!DOCTYPE html>
<html>

<head>
<style>
ul.a {
        list-style: square;
        background: pink;
        padding: 20px;
      }
</style>
</head>

<body>
<h2>
      IFT 203 Class
</h2>
<p> Unordered lists </p>
<ul class="a">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
</body>

</html>
```

3.8 CSS Tables

A **table** in CSS is used to apply the various styling properties to the HTML Table elements to arrange the data in rows and columns, or possibly in a more complex structure in a properly organized manner. Tables are widely used in communication, research, and data analysis. The table-layout property in CSS can be utilized to display the layout of the table. This property is basically used to sets the algorithm that is used to layout <table>cells, rows, and columns.

Properties:

Border: It is used for specifying borders in the table.

Syntax:

border: table_width table_color;

Example 1: This example describes the CSS Table to apply the border property.

```
<!DOCTYPE html>
<html>
```

```
<head>
<style>
   body {
      text-align: left;
   }

   h1 {
      color: green;
   }

   table,
   th,
   td {

      /* Styling the border. */
      border: 1.5px solid blue;
   }
</style>
</head>

<body>
<h1> IFT 203 Class </h1>
<h2>Add border to table:</h2>
<table>
<tr>
<th>Roll No.</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
</body>

</html>
```

Border Collapse: The border-collapse property tells us whether the browser should control the appearance of the adjacent borders that touch each other or whether each cell should maintain its style.

Syntax:

border-collapse: collapse/separate;

Example 2: This example describes the CSS Table by applying the border-collapse property.

```
<!DOCTYPE html>
<html>

<head>
<style>
   body {
      text-align: left;
   }

   h1 {
      color: green;
   }

   table.one {

      /* Styling border collapse for table one. */
      border-collapse: collapse;
   }

   table.two {

      /* Styling border separate for table two. */
      border-collapse: separate;
   }

   table,
   td,
   th {
      border: 1.5px solid blue;
   }
</style>
</head>

<body>
<h1>IFT 203 Class</h1>
<h2>borders collapsed:</h2>
<table class="one">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
```

```
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
<br>
<br>
<h2>borders separated:</h2>
<table class="two">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
</body>

</html>
```

Border Spacing: This property specifies the space between the borders of the adjacent cells.

Syntax:

```
border-spacing: value;
```

Example 3: This example describes the CSS Table by applying the border-spacing property.

```
<!DOCTYPE html>
<html>

<head>
<style>
   body {
      text-align: left;
   }

   h1 {
      color: green;
   }

   table.one {
```

```
        border-collapse: separate;

        /* Styling the border-spacing
              between adjacent cells. */
        border-spacing: 10px;
    }

    table.two {
        border-collapse: separate;

        /* Styling the border-spacing
             between adjacent cells. */
        border-spacing: 10px 30px;
    }

    table,
    td,
    th {
        border: 1.5px solid blue;
    }
</style>
</head>

<body>
<h1> IFT 203 Class </h1>
<h2>border spacing:</h2>
<table class="one">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
<br>
<br>
<h2>border spacing:</h2>
<table class="two">
<tr>
<th>Roll Number</th>
```

```
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
</body>

</html>
```

Caption Side: Caption side property is used for controlling the placement of caption in the table. By default, captions are placed above the table.

Syntax:

```
caption-side: top/bottom;
```

Example 4: This example describes the CSS Table by applying the caption-side property to control the placement of the Table caption.

```
<!DOCTYPE html>
<html>

<head>
<style>
   body {
      text-align: left;
   }

   h1 {
      color: green;
   }

   table.one {
      border-collapse: separate;
      border-spacing: 10px;

      /* Controlling the placement of caption. */
      caption-side: top;
   }

   table.two {
      border-collapse: separate;
      border-spacing: 10px;
```

```
        /* Controlling the placement of caption. */
        caption-side: bottom;
    }

    table,
    td,
    th {
        border: 1.5px solid blue;
    }
</style>
</head>

<body>
<h1> IFT 203 Class</h1>
<h2>Caption on top:</h2>
<table class="one">
<caption>Caption at the top of the table.</caption>
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
<br>
<br>
<h2>Caption at bottom:</h2>
<table class="two">
<caption> Caption at the bottom of the table </caption>
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
```

```
</tr>
</table>
</body>

</html>
```

Empty cells: This property specifies whether or not to display borders and background on empty cells in a table.

Syntax:

```
empty-cells:show/hide;
```

Example 5: This example describes the CSS Table by applying the empty-cell property that specifies whether to display the borders or not in the empty cells in a table.

```
<!DOCTYPE html>
<html>

<head>
<style>
   body {
      text-align: left;
   }

   h1 {
      color: green;
   }

   table.one {
      border-collapse: separate;
      border-spacing: 10px;

      /* Hiding empty cells border */
      empty-cells: hide;
   }

   table.two {
      border-collapse: separate;
      border-spacing: 10px;

      /* Display empty cells border */
      empty-cells: show;
   }

   table,
   td,
   th {
      border: 1.5px solid blue;
   }
```

```
</style>
</head>

<body>
<h1> IFT 203 Class</h1>
<h2>empty cells hide:</h2>
<table class="one">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
<br>
<br>
<h2>empty cells show:</h2>
<table class="two">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
</body>

</html>
```

Table layout: The table layout property is used to set up the layout algorithm used for the table.

Syntax:

table-layout:auto/fixed;

Example 6: This example describes the CSS Table by applying the table layout property.

```
<!DOCTYPE html>
<html>

<head>
<style>
   body {
      text-align: left;
   }

   h1 {
      color: green;
   }

   table.one {
      width: 80px border-collapse: separate;
      border-spacing: 10px;

      /* Layout of table is auto. */
      table-layout: auto;
   }

   table.two {
      width: 80px border-collapse: separate;
      border-spacing: 10px;
      /* Layout of table is fixed. */
      table-layout: fixed;
   }

   table,
   td,
   th {
      border: 1.5px solid blue;
      width: 80px;
   }
</style>
</head>

<body>
<h1> IFT 203 Class</h1>
<h2>auto table layout:</h2>
<table class="one">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
```

```
<td>1</td>
<td>A_B_C_D_E_F_G_H_I_J_K_L_M_N_O_P</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
<br>
<br>
<h2>fixed table layout:</h2>
<table class="two">
<tr>
<th>Roll Number</th>
<th>Name</th>
</tr>
<tr>
<td>1</td>
<td>A_B_C_D_E_F_G_H_I_J_K_L_M_N_O_P</td>
</tr>
<tr>
<td>2</td>
<td>X_Y_Z</td>
</tr>
</table>
</body>

</html>
```

Self-Assessment Exercise(s)

(1) Which of the following is the correct syntax to link an external CSS file in an HTML document?

A. <link rel="stylesheet" type="text/css" href="styles.css">
B. <stylesheet>styles.css</stylesheet>
C. <css link="styles.css">
D. <style src="styles.css"></style>

Answer: A. <link rel="stylesheet" type="text/css" href="styles.css">

(2) What does the 'C' in CSS stand for?

A. Color
B. Code
C. Cascading
D. Common

Answer: C. Cascading

(3) Which property is used to change the background color of an element in CSS?
A. color
B. background
C. bgcolor
D. background-color
Answer: D. background-color

(4) How can you make a list that lists its items with squares in CSS?
A. list-type: square;
B. list-style-type: square;
C. list-style: square;
D. list-square: true;
Answer: B. list-style-type: square;

(5) Which of the following selectors is used to apply a style to a group of elements with the same class in CSS?
A. #classname
B. .classname
C. classname
D. *classname
Answer: B. .classname

Conclusion

This unit has equipped you with the foundational knowledge and skills necessary to effectively style and layout web pages. You have learned how to use selectors, properties, and values to apply various styles and understand the importance of the cascade, inheritance, and specificity in resolving conflicts between styles. By mastering these concepts, you are now able to create visually appealing and responsive designs that enhance the user experience. As you continue to explore more advanced CSS techniques and practices, this foundational understanding will serve as a crucial building block in your journey toward becoming a proficient web developer.

4.0 Summary

Introduction to CSS unit provides an essential overview of CSS, a key technology for styling web pages. It begins with the fundamental concept of separating content (HTML) from presentation (CSS), highlighting the benefits of this approach, such as easier maintenance

and greater flexibility. The unit covers basic syntax, including selectors, properties, and values, and demonstrates how to link CSS to HTML documents. It also introduces different types of selectors and explains the cascading and inheritance principles that govern how styles are applied. Through examples and exercises, learners gain practical skills in creating visually appealing and consistent web pages.

5.0 References/Further Readings

Lie, H. W., & Bos, B. (2005). Cascading style sheets: Designing for the web. Addison-Wesley Professional.

Schengili-Roberts, K. (2004). Core CSS: Cascading style sheets. Prentice Hall Professional.

Bartlett, J. (2022). Introduction to Cascading Style Sheets. In Programming for Absolute Beginners: Using the JavaScript Programming Language (pp. 91-103). Berkeley, CA: Apress.

Wilson, D., Hassan, S. U., Aljohani, N. R., Visvizi, A., & Nawaz, R. (2023). Demonstrating and negotiating the adoption of web design technologies: Cascading Style Sheets and the CSS Zen Garden. Internet Histories, 7(1), 27-46.

Donahoe, L., & Hartl, M. (2022). Learn Enough HTML, CSS and Layout to be Dangerous: An Introduction to Modern Website Creation and Templating Systems. Addison-Wesley Professional.

Casabona, J. (2020). HTML and CSS: Visual QuickStart Guide. Peachpit Press.

Unit 2 Styling with Cascading Style Sheet

Contents

1.0 Introduction

CSS is a fundamental skill in web development that transforms basic HTML documents into visually appealing and well-structured web pages. CSS is a style sheet language that allows developers to control the layout, design, and presentation of web content. It works by applying styles to HTML elements using a variety of selectors and properties, enabling precise control over aspects such as colors, fonts, spacing, and overall page layout. By separating content (HTML) from presentation (CSS), developers can maintain cleaner and more manageable code, enhance the user experience, and ensure a consistent look and feel across multiple web pages.This unit on Styling with CSS will cover essential concepts and techniques, starting with the basics of including CSS in HTML documents through inline, internal, and external styles. Learners will explore the various types of selectors, including class, ID, and pseudo-selectors, to target specific elements for styling. Key CSS properties for layout control, such as display, position, and float, will be examined, along with techniques for creating responsive designs that adapt to different screen sizes and devices. By the end of this unit, students will be equipped with the knowledge and skills to create well-designed, user-friendly web pages that leverage the full power of CSS.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss the CSS selectors and style rules
- explain margins and padding
- discuss Positioning HTML elements with CSS
- explain CSS media queries and responsive design
- explain alignment and text transformations

3.0 Main Content

3.1 CSS selectors and style rules

The CSS standard defines selectors and style rules. The syntax is defined as follows:
selector { property:value; }
A selector can be one of a predefined identifier (, e.g., H1), a class name (e.g. .myclass), or an identifier (e.g. #myuniqueid).
In CSS an identifier is supposed to be unique across all of the elements in a page (or window in our case) while a class can be assigned to several elements.
For example, the following CSS file defines the size and color of the 'h1' tag.
h1 { color:red; font-size:48px; }
CSS pseudo classes
CSS pseudo-classes are used to qualify attributes of selectors. For example, you can select a visited link in HTML and the style is different.
a:visited { color:red; }
2. Exercise: Style an HTML page with CSS
In the following you create a local html page and style it with CSS. Create a new directory and create the following file called 'styles.css'.

```
/* this is a comment */
/* we style only the h1 element*/

h1 {
   border-style:solid none solid solid;
color:red;
}
```

In the same directory define the following HTML file. This file defines that it uses the 'styles.css' style sheet file from the same same directory.

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML5 Document</title>
<link href="styles.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1>Your first HTML5 page</h1>
<p>This is a <a href="http://www.vogella.com">link</a> to another webpage</p>
<!-- this is a comment -->
</body>
</html>
```

3. HTML container via id and class

HTML allows to define sections via 'div' containers. These 'div' containers can be used to style parts of the HTML document differently. For this purpose you can identify the div containers via id or class attributes.

While id and class generate the same output, an id must be unique in the HTML document while the class attribute can be defined for several HTML elements in a page. CSS allows to style these elements via special selectors.

The following rule applies:

Styling div containers

Definition	**CSS selection rule**
Content	#myid {css rules….}
Content	.myclass {css rules…}

The following example demonstrate both usages. Create the file stylesdiv.css

```
/* this is a comment */
/* we style only the h1 element*/

#number1 {
color:red;
}

#number2 {
color:blue;
}
```

```
.class1 {
   font-weight: bold;
}

.class2{
   font-weight: normal;
   color: green;
}
```

Create the following HTML file to use the style sheet.

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML5 Document</title>
<link href="stylesdiv.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="number1"> Some Text </div>
<div id="number2"> Another Text</div>
<div class="class1"> Styling with classes </div>
<div class="class2"> Another class </div>
</body>
</html>
```

You can also select by position in the HTML document. For example 'td a' only selects links which are within a table row.

You can also use other attributes for example the following will define certain styling for links which have been visited or over which the mouse hovers. They will identify if you have a link already visited or if the mouse hovers over a link and will change the display of the link accordingly.

```
a:visited {color:grey}
a:hover {text-decoraton:underline}
```

CSS includes

A CSS file can import other CSS files via the '@import' statement. It must be the first rule in the style sheet using the '@import' statement.

```
@import "mystyle.css";
@import url("mystyle.css");
```

If you want to import a css file from a html file it is better to use the following statement:

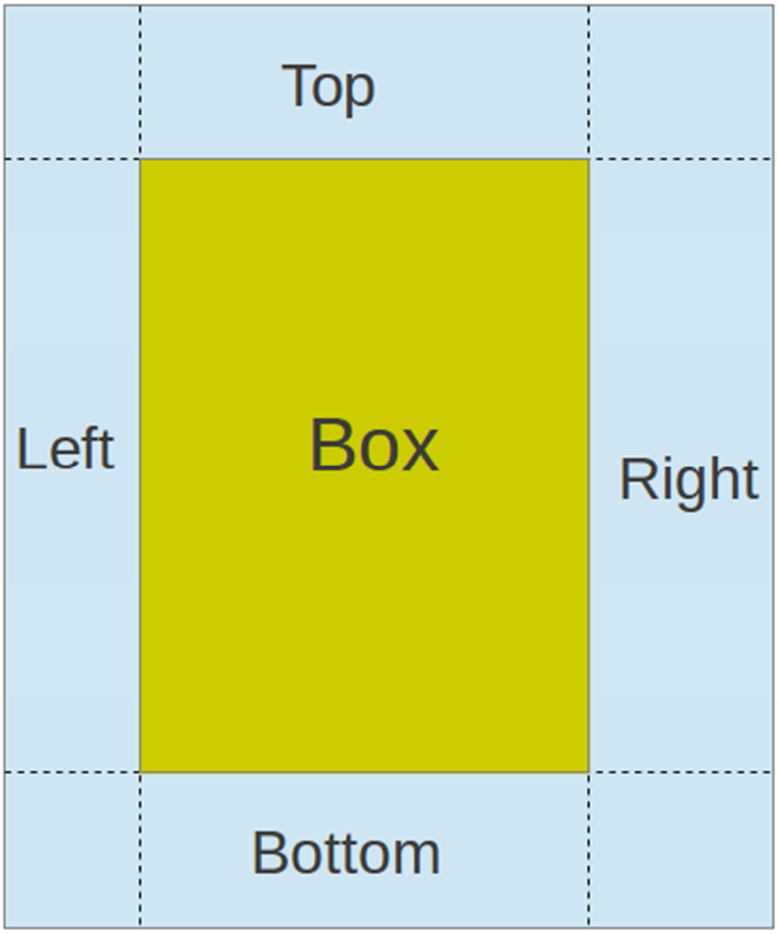
```
<link rel="stylesheet" href="include.css">
```

and not

```
@import "mystyle.css";
        @import url("mystyle.css");
```

3.2 Margins and padding

Margins
A block element can be thought of as a box that contains something. This box has a border to other elements and you can influence the distance to other elements via the 'margin' and 'padding' settings.
Margin defines the outer distance of other elements. You can set values for top, right, bottom, and left.



You can define the margins for a box individually or combine them into one statement.

```
body {
   margin-top: 10px;
   margin-right: 120px;
   margin-bottom: 20px;
   margin-left: 8px;
}
body {
   margin: 10px 120px 20px 8px;
}
```

Padding

Padding defines the inner distance of elements to the end of the box.

```
<!DOCTYPE html>
<html>
<head>
<title>Margin, padding test</title>
<link href="styles.css" rel="stylesheet" type="text/css">
</head>

<body>
<p id="test">Your first HTML5 page</p>
</body>
</html>
#test {
   width: 200px;
   height: 60px;
   padding: 10px;
   border: 5px dotted blue;
   margin: 5px, 10px, 20px, 30px;
}
```

The total size of the HTML box is defined by the initial size of the box, plus the margins the padding, and a border, if defined.

3.3 Positioning HTML elements with CSS

CSS allows to setup of elements with fixed, related, and absolute positions. Relative is the standard and will change the distribution of the different text containers based on the available space.

Frequently you want to make sure that you boxes stay on a specific place. You can use postion:absolute for this. A block element with this style will be removed from the normal flow of the HTML page and will have a fixed space. For example:

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML5 Document</title>
<style type="text/css">
#center {
   position: absolute;
   width: 200px;
   left: 400px;
   background-color: green;
}
#left {
   position: absolute;
```

```
   width: 200px;
   background-color: blue;
   top:200px;
}
#right {
   position: absolute;
   background-color: red;
   left: 200px;
   width: 200px;
}
</style>
</head>
<body>
<h1>HTML5 with CSS positioning</h1>
<p id="center">Center fixed box</p>
<p id="left">Left fixed box</p>
<p id="right">Right fixed box</p>

</body>
</html>
```

If you want to have a element always on a certain position you can use the fixed position and will not move even if you scroll down the HTML page.

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML5 Document</title>
<style type="text/css">
p.position_fixed {
   position: fixed;
   top: 200px;
   right: 5px;
}
</style>
</head>
<body>
<h1>HTML5 with CSS positioning</h1>
<p class="position_fixed"><a
href="http://www.twitter.com/vogella">Follow
vogella on twitter</a></p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
```

```
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>
<p>lots of information here</p>

</body>
</html>
```

CSS-based layout

You can use HTML div container and CSS to layout your webpage. Your example the following webpage uses div container.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>This is a tile</title>
<link href="styleslayout.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id="header">Header Section</div>
<div id="leftcol">Left Section</div>
<div id="rightcol">Right Section</div>
<div id="content">Content Section</div>
<div id="footer">Footer Section</div>
</body>
</html>
<style type ="text/css">

body {
   margin: 0px;
   padding: 0px;
}

#header {
   background: #aba;
   width: 100%;
}

#leftcol {
   background: #aaa;
   float: left;
   width: 20%;
```

```
    height: 500px;
}

#rightcol {
    background: #aaa;
    float: right;
    width: 20%;
    height: 500px;
}

#content {
    background: #eee;
    float: left;
    width: 59%;
    height: 500px;
}

#footer {
    background: #aba;
    clear: both;
    width: 100%;
}
</style>
```

This result in the following layout.

3.4 CSS media queries and responsive design

Via CSS you can use media queries to define CSS settings based on certain criteria. A common use case is to have different CSS based styling for devices with only limited pixels and other designs for larger screens.

For example the following CSS defines a fixed position for a search box. If the screen has a maximum width of 750 or less different styling is used.

```
#searchfixed {
```

```
    position: fixed;
    top: 8px;
    right: 200px;
    z-index: 4;
    width:200px;
}

#searchwrapper{
    width:100%;
    height:40px;
position:relative;
}

#search_field {
    margin-left:40px;
    height:20px;
    width:150px;
border:none;
    background-color:#fcfcfc;
    border-radius:5px;
    -moz-border-radius:5px;
    -webkit-border-radius:5px;
    padding-left: 5px;
}

#search_button {
position:absolute;
    top:3px;
    right:15px;
}

@media screen and (max-width: 750px) {

    #topnav {
        height: 80px;
    }

    #topnav ul {
        width: 100%;
    }

    #logo {
        top: 30px !important;
    }

    #searchfixed {
```

```
    top: 45px !important;
    left: 0;
    right: auto !important;
}

#search_button {
    right: 45px !important;
}

#search_field {
    margin-left: 0.55em;
}

#container, #trainingcontainer {
    margin: 90px auto !important;
}

}
```

Defining size

The most consist way to define size is the unit 'em' which is a relative unit to the font-size. 1em is as large as the font size. You can use 1em for defining the space between words (word-spacing), or letters (letter-spacing) or to define the line height (line height) of an HTML element. text-indent allows you to define the intent of the first line of a paragraph. text-indent: -1em put the first line a bit before the rest of the text.

3.5 Alignment and Text transformations

Via text-align, you can define how your content (not only text) should be aligned. text-transform allows to transform the text to upper, or lower case or to capitalize the first letter.

```
text-transform:uppercase;
text-transform:lowercase;
text-transform:capitalize;

text-align:left;
text-align:right;
text-align:center;
text-align:justify;
```

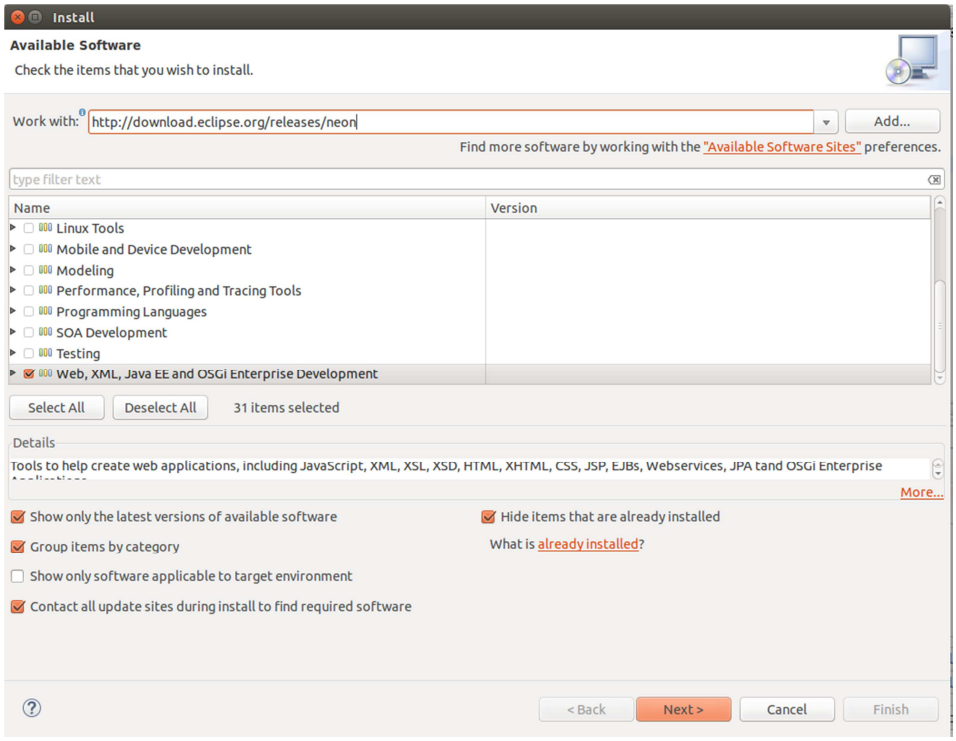
Versioning of CSS

The versioning of css files is not mandatory, but it could be very useful to force the browser to load a changed css file instead of using the CSS caching.

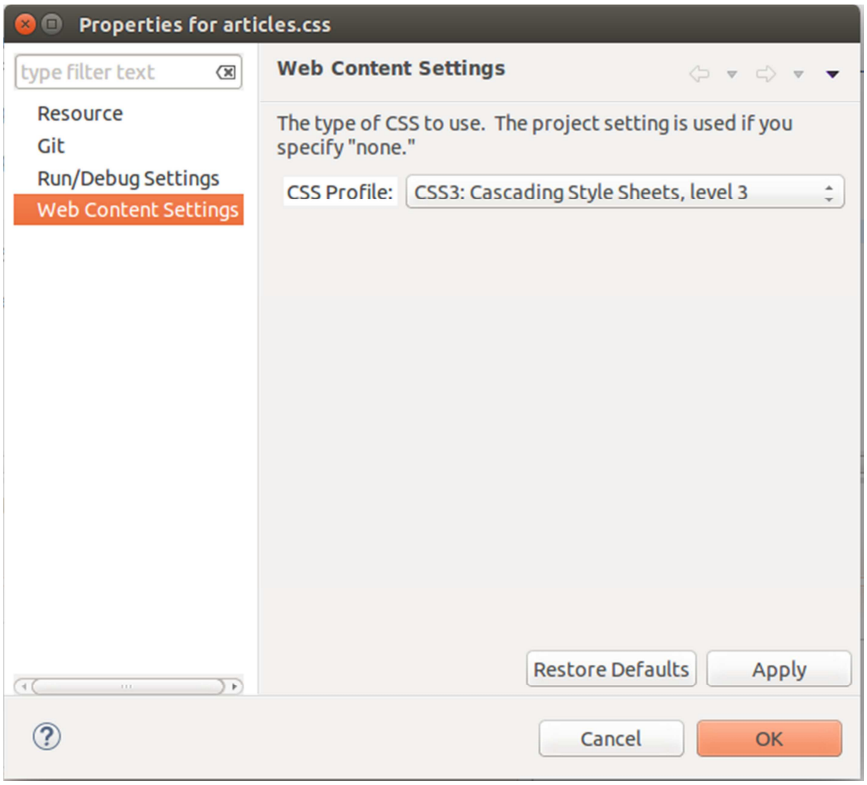
To achieve this just add ?version=1.1 at the end of the css file name when referencing to the file.
So instead of calling a css file like this:
https://www.vogella.com/css/companyfooter.css
you should replace it with
https://www.vogella.com/css/companyfooter.css?version=1.1
The wording after the '?' has no relevance and can be chosen freely. With this solution, your css changes are immediately visible and you do not need to refresh your browser with CTRL F5 until you see them. Additionally the name of the css file keeps the same which saves you a global search and replacement.

CSS editor usage in Eclipse

Eclipse has an integrated CSS editor which by default supports CSS2. To see the new CSS3 properties you must activate this feature.
You can activate it per file, Dynamic Web Project or Static Web Project, but not for other types of Projects. It is also not available as a workspace-wide preference.
First, just make sure you have installed the Web Package of Eclipse. Otherwise, you won't see the Web Content Settings in the Properties page. In Eclipse go to Help → Install New Software … → select your Eclipse release software site → select the Web Package → Press Next >



After the installation perform the following steps:
Select a CSS file, Dynamic Web Project or Static Web Project and right-click on it
Click on Properties
Select the Web Content Settings
Set the CSS Profile value to "CSS3: Cascading Style Sheets, level 3"
Click the OK button.



This guide applies to Eclipse releases up to Mars. With Neon M6 the default CSS profile will be set to CSS3.

Self-Assessment Exercise(s)

(1) Which CSS property is used to change the text color of an element?
a) font-color
b) text-color
c) color
d) text-style
Answer: c) color

(2) How do you make each word in a text start with a capital letter using CSS?
a) text-transform: uppercase;
b) text-transform: lowercase;

c) text-transform: capitalize;
d) text-transform: initial;
Answer: c) text-transform: capitalize;

(3) Which CSS property is used to change the background color of an element?
a) background-color
b) color
c) bg-color
d) background-style
Answer: a) background-color

(4) How do you select an element with the id "header" in CSS?
a) .header
b) header
c) #header
d) *header
Answer: c) #header

(5) Which property is used to control the space between lines of text?
a) line-height
b) letter-spacing
c) text-spacing
d) word-spacing
Answer: a) line-height

Conclusion

CSS is essential for web development, enabling designers to enhance the visual appeal and user experience of websites effectively. Through the comprehensive exploration of CSS syntax, selectors, properties, and responsive design principles, learners can create aesthetically pleasing, dynamic, and accessible web pages. By understanding the cascade, inheritance, and specificity, developers can efficiently manage complex styles across multiple pages. Ultimately, proficiency in CSS not only empowers developers to implement sophisticated design elements but also lays a solid foundation for advanced web technologies and frameworks.

4.0 Summary

CSS is a fundamental aspect of web development, enabling designers to control the visual presentation of web pages. CSS allows for the separation of content HTML from design, providing a more flexible

and efficient method for styling web pages. By using CSS, developers can apply styles such as colors, fonts, spacing, and layout properties to HTML elements, ensuring a consistent look across multiple pages of a website. This separation not only enhances maintainability but also improves the user experience by allowing for faster page loading times and easier adjustments to the site's design.CSS operates on a rule-based system where selectors target HTML elements, and declarations within curly braces define the style properties to be applied. These rules can be applied inline, embedded within the HTML file using the <style> tag, or in external style sheets linked via the <link> tag. The cascading nature of CSS means that styles can be overridden by more specific rules or those defined later in the stylesheet, allowing for sophisticated and nuanced design control. Advanced features such as media queries enable responsive design, adapting the layout to different screen sizes and devices, while CSS frameworks like Bootstrap provide pre-designed components and grid systems to streamline the development process.

5.0 References/Further Reading

Robbins, J. N. (2012). Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. " O'Reilly Media, Inc.".

McGrath, M. (2020). HTML in easy steps. In Easy Steps.

Tabarés, R. (2021). HTML5 and the evolution of HTML; tracing the origins of digital platforms. Technology in Society, 65, 101529.

Macaulay, M. (2017). Introduction to web interaction design: With Html and Css. Chapman and Hall/CRC.

Rebah, H. B., Boukthir, H., & Chedebois, A. (2022). Website Design and Development with HTML5 and CSS3. John Wiley & Sons.

MODULE 4 INTRODUCTION TO JAVASCRIPT

MODULE INTRODUCTION

Introduction to JavaScript a foundational module designed to equip you with the essential skills and knowledge to start coding in JavaScript. JavaScript is a versatile, high-level programming language that enables developers to create dynamic and interactive web applications. This module will guide you through the basics, laying a solid foundation for your journey into web development. From simple form validations to complex animations and single-page applications, JavaScript is the backbone of modern web development. This module is designed to guide you through the fundamental concepts and syntax of JavaScript, making it accessible even if you have no prior programming experience. By the end of this course, you will have the skills to write and understand basic JavaScript code and apply it to enhance the functionality of web pages.Throughout this module, you will explore a variety of essential topics, including variables, data types, operators, control structures, functions, objects, and arrays. Additionally, you will learn how to manipulate the Document Object Model (DOM) to create dynamic and interactive web pages. The module also covers debugging techniques and best practices to help you write clean, efficient, and maintainable code. With a mix of theoretical knowledge and practical assignments, this module aims to build a solid foundation in JavaScript, preparing you for more advanced studies in web development and beyond. Get ready to dive into coding and transform your ideas into interactive web applications.

Unit 1 Basics of JavaScript

Contents

1.0 Introduction

JavaScript is a versatile and powerful programming language that is essential for creating dynamic and interactive web content. Unlike HTML and CSS, which structure and style web pages respectively, JavaScript adds behavior to web pages, making it possible to create features such as interactive forms, animations, and real-time content updates. This unit on the Basics of JavaScript will introduce you to the fundamental concepts and syntax of the language, providing a solid foundation for understanding how JavaScript can be used to enhance web development projects. We will cover essential topics including variables, data types, operators, control structures, functions, and events, each of which plays a crucial role in building interactive web applications. By the end of this unit, you will have the knowledge and skills to write basic JavaScript programs and integrate them into web pages to create interactive user experiences.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss the Code structure
- explain Comments
- discuss Variables and data types
- explain interaction: alert, prompt, confirm
- explain basic operators, maths

3.0 Main Content

3.1 Introduction to JavaScript

Let's see what's so special about JavaScript, what we can achieve with it, and what other technologies play well with it.

JavaScript was initially created to "make web pages alive".The programs in this language are called scripts. They can be written right in a web page's HTML and run automatically as the page loads.Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.In this aspect, JavaScript is very different from another language called Java.

When JavaScript was created, it initially had another name: "LiveScript". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help.

But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

The browser has an embedded engine sometimes called a "JavaScript virtual machine".

Different engines have different "codenames". For example:
V8 – in Chrome, Opera, and Edge.

Spider Monkey – in Firefox.
…There are other codenames like "Chakra" for IE, "JavaScriptCore", "Nitro" and "SquirrelFish" for Safari, etc.

The terms above are good to remember because they are used in developer articles on the internet. We'll use them too. For instance, if "a feature X is supported by V8", then it probably works in Chrome, Opera and Edge.

How do engines work?
Engines are complicated. But the basics are easy.

The engine (embedded if it's a browser) reads ("parses") the script.
Then it converts ("compiles") the script to machine code.
And then the machine code runs, pretty fast.

The engine applies optimizations at each step of the process. It even watches the compiled script as it runs, analyzes the data that flows through it, and further optimizes the machine code based on that knowledge.

What can in-browser JavaScript do?
Modern JavaScript is a "safe" programming language. It does not provide low-level access to memory or the CPU, because it was initially created for browsers which do not require it.
JavaScript's capabilities greatly depend on the environment it's running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.
In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.

For instance, in-browser JavaScript is able to:
Add new HTML to the page, change the existing content, modify styles.
React to user actions, run on mouse clicks, pointer movements, key presses.

Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).

Get and set cookies, ask questions to the visitor, show messages.
Remember the data on the client-side ("local storage").
What CAN'T in-browser JavaScript do?

JavaScript's abilities in the browser are limited to protect the user's safety. The aim is to prevent an evil webpage from accessing private information or harming the user's data.

Examples of such restrictions include:
JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like "dropping" a file into a browser window or selecting it via an <input> tag.

There are ways to interact with the camera/microphone and other devices, but they require a user's explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the NSA.

Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other page if they come from different sites (from a different domain, protocol or port).

This is called the "Same Origin Policy". To work around that, both pages must agree for data exchange and must contain special JavaScript code that handles it. We'll cover that in the tutorial.

This limitation is, again, for the user's safety. A page from http://anysite.com which a user has opened must not be able to access another browser tab with the URL http://gmail.com, for example, and steal information from there.

JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that's a safety limitation.

Such limitations do not exist if JavaScript is used outside of the browser, for example on a server. Modern browsers also allow plugins/extensions which may ask for extended permissions.

What makes JavaScript unique?

There are at least three great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Supported by all major browsers and enabled by default.
  JavaScript is the only browser technology that combines these three things.

That's what makes JavaScript unique. That's why it's the most widespread tool for creating browser interfaces.
That said, JavaScript can be used to create servers, mobile applications, etc.

Languages "over" JavaScript

The syntax of JavaScript does not suit everyone's needs. Different people want different features.

That's to be expected because projects and requirements are different for everyone.

So, recently a plethora of new languages appeared, which are transpiled (converted) to JavaScript before they run in the browser.
Modern tools make the transpilation very fast and transparent, actually allowing developers to code in another language and auto-converting it "under the hood".

Examples of such languages:

- Coffee Script is "syntactic sugar" for JavaScript. It introduces shorter syntax, allowing us to write clearer and more precise code. Usually, Ruby devs like it.
- TypeScript is concentrated on adding "strict data typing" to simplify the development and support of complex systems. It is developed by Microsoft.
- Flow also adds data typing but in a different way. Developed by Facebook.
- Dart is a standalone language that has its engine that runs in non-browser environments (like mobile apps), but also can be transpiled to JavaScript. Developed by Google.
- Brython is a Python transpiler to JavaScript that enables the writing of applications in pure Python without JavaScript.
- Kotlin is a modern, concise, and safe programming language that can target the browser or Node.

There are more. Of course, even if we use one of these transpired languages, we should also know JavaScript to understand what we're doing.

3.2 Code structure

The first thing we'll study is the building blocks of code.

Statements
Statements are syntax constructs and commands that perform actions.
We've already seen a statement, alert('Hello, world!'), which shows the message "Hello, world!".

We can have as many statements in our code as we want. Statements can be separated with a semicolon.
For example, here we split "Hello World" into two alerts:

```
alert('Hello'); alert('World');
```

Usually, statements are written on separate lines to make the code more readable:

```
alert('Hello');
```

alert('World');
Semicolons
A semicolon may be omitted in most cases when a line break exists.
This would also work:
alert('Hello')
alert('World')
Here, JavaScript interprets the line break as an “implicit” semicolon. This is called an automatic semicolon insertion.

In most cases, a newline implies a semicolon. But “in most cases” does not mean “always”!
There are cases when a newline does not mean a semicolon. For example:
alert(3 +
1
+ 2);
The code outputs 6 because JavaScript does not insert semicolons here. It is intuitively obvious that if the line ends with a plus "+", then it is an “incomplete expression”, so a semicolon there would be incorrect. And in this case, that works as intended.

But there are situations where JavaScript “fails” to assume a semicolon where it is really needed.
Errors which occur in such cases are quite hard to find and fix.
An example of an error
If you’re curious to see a concrete example of such an error, check this code out:
alert("Hello");

[1, 2].forEach(alert);
No need to think about the meaning of the brackets [] and forEach yet. We’ll study them later. For now, just remember the result of running the code: it shows Hello, then 1, then 2.
Now let’s remove the semicolon after the alert:
alert("Hello")

[1, 2].forEach(alert);
The difference compared to the code above is only one character: the semicolon at the end of the first line is gone.
If we run this code, only the first Hello shows (and there’s an error, you may need to open the console to see it). There are no numbers any more.
That’s because JavaScript does not assume a semicolon before square brackets [...]. So, the code in the last example is treated as a single statement.
Here’s how the engine sees it:

```
alert("Hello")[1, 2].forEach(alert);
```

Looks weird, right? Such merging in this case is just wrong. We need to put a semicolon after alert for the code to work correctly.

This can happen in other situations also.

We recommend putting semicolons between statements even if they are separated by newlines. This rule is widely adopted by the community. Let's note once again – *it is possible* to leave out semicolons most of the time. But it's safer – especially for a beginner – to use them.

3.3 Comments

As time goes on, programs become more and more complex. It becomes necessary to add *comments* which describe what the code does and why.

Comments can be put into any place of a script. They don't affect its execution because the engine simply ignores them.

One-line comments start with two forward slash characters //.

The rest of the line is a comment. It may occupy a full line of its own or follow a statement.

Like here:

```
// This comment occupies a line of its own
alert('Hello');

alert('World'); // This comment follows the statement
```

Multiline comments start with a forward slash and an asterisk /* and end with an asterisk and a forward slash */.

Like this:

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

The content of comments is ignored, so if we put code inside /* … */, it won't execute.

Sometimes it can be handy to temporarily disable a part of code:

```
/* Commenting out the code
alert('Hello');
*/
alert('World');
```

Use hotkeys!

In most editors, a line of code can be commented out by pressing the Ctrl+/ hotkey for a single-line comment and something like Ctrl+Shift+/ – for multiline comments (select a piece of code and press the hotkey). For Mac, try Cmd instead of Ctrl and Option instead of Shift.

Nested comments are not supported!

There may not be /*...*/ inside another /*...*/.

Such code will die with an error:

```
/*
  /* nested comment ?!? */
*/
alert( 'World' );
```

Please, don't hesitate to comment on your code.

Comments increase the overall code footprint, but that's not a problem at all. Many tools minify code before publishing it to a production server. They remove comments, so they don't appear in the working scripts. Therefore, comments do not have negative effects on production at all.

3.4 Variables

Most of the time, a JavaScript application needs to work with information. Here are two examples:

An online shop – the information might include goods being sold and a shopping cart.

A chat application – the information might include users, messages, and much more.

Variables are used to store this information.

A variable

A variable is a "named storage" for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the let keyword.

The statement below creates (in other words: *declares*) a variable with the name "message":

```
let message;
```

Now, we can put some data into it by using the assignment operator =:

```
let message;

message = 'Hello'; // store the string 'Hello' in the variable named message
```

The string is now saved into the memory area associated with the variable. We can access it using the variable name:

```
let message;
message = 'Hello!';

alert(message); // shows the variable content
```

To be concise, we can combine the variable declaration and assignment into a single line:

```
let message = 'Hello!'; // define the variable and assign the value
alert(message); // Hello!
```

We can also declare multiple variables in one line:

```
let user = 'John', age = 25, message = 'Hello';
```

That might seem shorter, but we don't recommend it. For the sake of better readability, please use a single line per variable.
The multiline variant is a bit longer, but easier to read:

```
let user = 'John';
let age = 25;
let message = 'Hello';
```

Some people also define multiple variables in this multiline style:

```
let user = 'John',
  age = 25,
  message = 'Hello';
```

…Or even in the "comma-first" style:

```
let user = 'John'
  , age = 25
  , message = 'Hello';
```

Technically, all these variants do the same thing. So, it's a matter of personal taste and aesthetics.
var instead of let
In older scripts, you may also find another keyword: var instead of let:

```
var message = 'Hello';
```

The var keyword is *almost* the same as let. It also declares a variable but in a slightly different, "old-school" way.
There are subtle differences between let and var, but they do not matter to us yet. We'll cover them in detail in the chapter The old "var".
A real-life analogy
We can easily grasp the concept of a "variable" if we imagine it as a "box" for data, with a uniquely-named sticker on it.
For instance, the variable message can be imagined as a box labelled "message" with the value "Hello!" in it:
We can put any value in the box.
We can also change it as many times as we want:

```
let message;

message = 'Hello!';

message = 'World!'; // value changed

alert(message);
```

When the value is changed, the old data is removed from the variable:
We can also declare two variables and copy data from one into the other.

```
let hello = 'Hello world!';

let message;

// copy 'Hello world' from hello into message
message = hello;
```

```
// now two variables hold the same data
alert(hello); // Hello world!
alert(message); // Hello world!
```

Declaring twice triggers an error

A variable should be declared only once.

A repeated declaration of the same variable is an error:

```
let message = "This";

// repeated 'let' leads to an error
let message = "That"; // SyntaxError: 'message' has already been declared
```

So, we should declare a variable once and then refer to it without let.

Functional languages

It's interesting to note that there exist so-called pure functional programming languages, such as Haskell, that forbid changing variable values.

In such languages, once the value is stored "in the box", it's there forever. If we need to store something else, the language forces us to create a new box (declare a new variable). We can't reuse the old one.

Though it may seem a little odd at first sight, these languages are quite capable of serious development. More than that, there are areas like parallel computations where this limitation confers certain benefits.

Variable naming

There are two limitations on variable names in JavaScript:

The name must contain only letters, digits, or the symbols $ and _.

The first character must not be a digit.

Examples of valid names:

```
let userName;
let test123;
```

When the name contains multiple words, camelCase is commonly used. That is: words go one after another, each word except first starting with a capital letter: myVeryLongName.

What's interesting – the dollar sign '$' and the underscore '_' can also be used in names. They are regular symbols, just like letters, without any special meaning.

These names are valid:

```
let $ = 1; // declared a variable with the name "$"
let _ = 2; // and now a variable with the name "_"

alert($ + _); // 3
```

Examples of incorrect variable names:

```
let 1a; // cannot start with a digit

let my-name; // hyphens '-' aren't allowed in the name
```

Case matters

Variables named apple and APPLE are two different variables.
Non-Latin letters are allowed, but not recommended
It is possible to use any language, including Cyrillic letters, Chinese logograms and so on, like this:

```
let имя = '...';
let 我 = '...';
```

Technically, there is no error here. Such names are allowed, but there is an international convention to use English in variable names. Even if we're writing a small script, it may have a long life ahead. People from other countries may need to read it sometime.
Reserved names
There is a list of reserved words, which cannot be used as variable names because they are used by the language itself.
For example: let, class, return, and function are reserved.
The code below gives a syntax error:

```
let let = 5; // can't name a variable "let", error!
let return = 5; // also can't name it "return", error!
```

An assignment without use strict
Normally, we need to define a variable before using it. But in the old times, it was technically possible to create a variable by a mere assignment of the value without using let. This still works now if we don't put use strict in our scripts to maintain compatibility with old scripts.

```
// note: no "use strict" in this example

num = 5; // the variable "num" is created if it didn't exist

alert(num); // 5
```

This is a bad practice and would cause an error in strict mode:

```
"use strict";

num = 5; // error: num is not defined
```

Constants
To declare a constant (unchanging) variable, use const instead of let:

```
const myBirthday = '18.04.1982';
```

Variables declared using const are called "constants". They cannot be reassigned. An attempt to do so would cause an error:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

When a programmer is sure that a variable will never change, they can declare it with const to guarantee and communicate that fact to everyone.
Uppercase constants
There is a widespread practice to use constants as aliases for difficult-to-remember values that are known before execution.

Such constants are named using capital letters and underscores.
For instance, let's make constants for colors in so-called "web" (hexadecimal) format:

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Benefits:
COLOR_ORANGE is much easier to remember than "#FF7F00".
It is much easier to mistype "#FF7F00" than COLOR_ORANGE.
When reading the code, COLOR_ORANGE is much more meaningful than #FF7F00.
When should we use capitals for a constant and when should we name it normally? Let's make that clear.
Being a "constant" just means that a variable's value never changes. But some constants are known before execution (like a hexadecimal value for red) and some constants are *calculated* in run-time, during the execution, but do not change after their initial assignment.
For instance:

```
const pageLoadTime = /* time taken by a webpage to load */;
```

The value of pageLoadTime is not known before the page load, so it's named normally. But it's still a constant because it doesn't change after the assignment.
In other words, capital-named constants are only used as aliases for "hard-coded" values.
Name things right
Talking about variables, there's one more extremely important thing.
A variable name should have a clean, obvious meaning, describing the data that it stores.
Variable naming is one of the most important and complex skills in programming. A glance at variable names can reveal which code was written by a beginner versus an experienced developer.
In a real project, most of the time is spent modifying and extending an existing code base rather than writing something completely separate from scratch. When we return to some code after doing something else for a while, it's much easier to find information that is well-labelled. Or, in other words, when the variables have good names.
Please spend time thinking about the right name for a variable before declaring it. Doing so will repay you handsomely.
Some good-to-follow rules are:
Use human-readable names like userName or shoppingCart.

Stay away from abbreviations or short names like a, b, and c, unless you know what you're doing.
Make names maximally descriptive and concise. Examples of bad names are data and value. Such names say nothing. It's only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
Agree on terms within your team and in your mind. If a site visitor is called a "user" then we should name related variables currentUser or newUser instead
of currentVisitor or newManInTown.
Sounds simple? Indeed it is, but creating descriptive and concise variable names in practice is not. Go for it.
Reuse or create?
And the last note. There are some lazy programmers who, instead of declaring new variables, tend to reuse existing ones.
As a result, their variables are like boxes into which people throw different things without changing their stickers. What's inside the box now? Who knows? We need to come closer and check.
Such programmers save a little bit on variable declaration but lose ten times more on debugging.
An extra variable is good, not evil.
Modern JavaScript minifiers and browsers optimize code well enough, so it won't create performance issues. Using different variables for different values can even help the engine optimize your code.
Summary
We can declare variables to store data by using the var, let, or const keywords.
let – is a modern variable declaration.
var – is an old-school variable declaration. Normally we don't use it at all, but we'll cover subtle differences from let in the chapter The old "var", just in case you need them.
const – is like let, but the value of the variable can't be changed.
Variables should be named in a way that allows us to easily understand what's inside them.
Tasks
Working with variables
importance: 2
Declare two variables: admin and name.
Assign the value "John" to name.
Copy the value from name to admin.
Show the value of admin using alert (must output "John").
solution
Giving the right name
importance: 3
Create a variable with the name of our planet. How would you name such a variable?

Create a variable to store the name of a current visitor to a website. How would you name that variable?
solution
Uppercase const?
importance: 4
Examine the following code:
const birthday = '18.04.1982';
const age = someCode(birthday);
Here we have a constant birthday for the date, and also the age constant.
The age is calculated from birthday using someCode(), which means a function call that we didn't explain yet (we will soon!), but the details don't matter here, the point is that age is calculated somehow based on the birthday.
Would it be right to use upper case for a birthday? For age? Or even for both?
const BIRTHDAY = '18.04.1982'; // make birthday uppercase?

const AGE = someCode(BIRTHDAY); // make age uppercase?

3.5 Data types

A value in JavaScript is always of a certain type. For example, a string or a number.
There are eight basic data types in JavaScript. Here, we'll cover them in general and in the next chapters we'll talk about each of them in detail.
We can put any type in a variable. For example, a variable can at one moment be a string and then store a number:
// no error
let message = "hello";
message = 123456;
Programming languages that allow such things, such as JavaScript, are called "dynamically typed", meaning that there exist data types, but variables are not bound to any of them.
Number
let n = 123;
n = 12.345;
The *number* type represents both integer and floating point numbers.
There are many operations for numbers, e.g. multiplication *, division /, addition +, subtraction -, and so on.
Besides regular numbers, there are so-called "special numeric values" which also belong to this data type: Infinity, -Infinity and NaN.
Infinity represents the mathematical Infinity ∞. It is a special value that's greater than any number.
We can get it as a result of division by zero:

alert(1 / 0); // Infinity
Or just reference it directly:
alert(Infinity); // Infinity
NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:
alert("not a number" / 2); // NaN, such division is erroneous
NaN is sticky. Any further mathematical operation on NaN returns NaN:
alert(NaN + 1); // NaN
alert(3 * NaN); // NaN
alert("not a number" / 2 - 1); // NaN
So, if there's a NaN somewhere in a mathematical expression, it propagates to the whole result (there's only one exception to that: NaN ** 0 is 1).
Mathematical operations are safe
Doing maths is "safe" in JavaScript. We can do anything: divide by zero, treat non-numeric strings as numbers, etc.
The script will never stop with a fatal error ("die"). At worst, we'll get NaN as the result.
Special numeric values formally belong to the "number" type. Of course, they are not numbers in the common sense of this word.
We'll see more about working with numbers in the chapter Numbers.
BigInt
In JavaScript, the "number" type cannot safely represent integer values larger than (253-1) (that's 9007199254740991), or less than -(253-1) for negatives.
To be really precise, the "number" type can store larger integers (up to 1.7976931348623157 * 10308), but outside of the safe integer range ±(253-1) there'll be a precision error, because not all digits fit into the fixed 64-bit storage. So an "approximate" value may be stored.
For example, these two numbers (right above the safe range) are the same:
console.log(9007199254740991 + 1); // 9007199254740992
console.log(9007199254740991 + 2); // 9007199254740992
So to say, all odd integers greater than (253-1) can't be stored at all in the "number" type.
For most purposes ±(253-1) range is quite enough, but sometimes we need the entire range of really big integers, e.g. for cryptography or microsecond-precision timestamps.
BigInt type was recently added to the language to represent integers of arbitrary length.
A BigInt value is created by appending n to the end of an integer:
// the "n" at the end means it's a BigInt
const bigInt = 1234567890123456789012345678901234567890n;

As BigInt numbers are rarely needed, we don't cover them here, but devoted them a separate chapter BigInt. Read it when you need such big numbers.

Compatibility issues

Right now, BigInt is supported in Firefox/Chrome/Edge/Safari, but not in IE.

String

A string in JavaScript must be surrounded by quotes.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

In JavaScript, there are 3 types of quotes.

Double quotes: "Hello".

Single quotes: 'Hello'.

Backticks: `Hello`.

Double and single quotes are "simple" quotes. There's practically no difference between them in JavaScript.

Backticks are "extended functionality" quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example:

```
let name = "John";

// embed a variable
alert( `Hello, ${name}!` ); // Hello, John!

// embed an expression
alert( `the result is ${1 + 2}` ); // the result is 3
```

The expression inside ${…} is evaluated and the result becomes a part of the string. We can put anything in there: a variable like name or an arithmetical expression like 1 + 2 or something more complex.

Please note that this can only be done in backticks. Other quotes don't have this embedding functionality!

```
alert( "the result is ${1 + 2}" ); // the result is ${1 + 2} (double quotes do nothing)
```

We'll cover strings more thoroughly in the chapter Strings.

There is no *character* type.

In some languages, there is a special "character" type for a single character. For example, in the C language and in Java it is called "char".

In JavaScript, there is no such type. There's only one type: string. A string may consist of zero characters (be empty), one character or many of them.

Boolean (logical type)

The boolean type has only two values: true and false.

This type is commonly used to store yes/no values: true means "yes, correct", and false means "no, incorrect".

For instance:
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
Boolean values also come as a result of comparisons:
let isGreater = 4 > 1;

alert(isGreater); // true (the comparison result is "yes")
We'll cover booleans more deeply in the chapter Logical operators.
The "null" value
The special null value does not belong to any of the types described above.
It forms a separate type of its own which contains only the null value:
let age = null;
In JavaScript, null is not a "reference to a non-existing object" or a "null pointer" like in some other languages.
It's just a special value which represents "nothing", "empty" or "value unknown".
The code above states that age is unknown.
The "undefined" value
The special value undefined also stands apart. It makes a type of its own, just like null.
The meaning of undefined is "value is not assigned".
If a variable is declared, but not assigned, then its value is undefined:
let age;

alert(age); // shows "undefined"
Technically, it is possible to explicitly assign undefined to a variable:
let age = 100;

// change the value to undefined
age = undefined;

alert(age); // "undefined"
…But we don't recommend doing that. Normally, one uses null to assign an "empty" or "unknown" value to a variable, while undefined is reserved as a default initial value for unassigned things.
Objects and Symbols
The object type is special.
All other types are called "primitive" because their values can contain only a single thing (be it a string or a number or whatever). In contrast, objects are used to store collections of data and more complex entities.
Being that important, objects deserve special treatment. We'll deal with them later in the chapter Objects after we learn more about primitives.

The symbol type is used to create unique identifiers for objects. We have to mention it here for the sake of completeness, but also postpone the details till we know objects.

The typeof operator

The typeof operator returns the type of the operand. It's useful when we want to process values of different types differently or just want to do a quick check.

A call to typeof x returns a string with the type name:

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

The last three lines may need additional explanation:

Math is a built-in object that provides mathematical operations. We will learn it in the chapter Numbers. Here, it serves just as an example of an object.

The result of typeof null is "object". That's an officially recognized error in typeof, coming from very early days of JavaScript and kept for compatibility. Definitely, null is not an object. It is a special value with a separate type of its own. The behavior of typeof is wrong here.

The result of typeof alert is "function", because alert is a function. We'll study functions in the next chapters where we'll also see that there's no special "function" type in JavaScript. Functions belong to the object type. But typeof treats them differently, returning "function". That also comes from the early days of JavaScript. Technically, such behavior isn't correct, but can be convenient in practice.

The typeof(x) syntax

You may also come across another syntax: typeof(x). It's the same as typeof x.

To put it clear: typeof is an operator, not a function. The parentheses here aren't a part of typeof. It's the kind of parentheses used for mathematical grouping.

Usually, such parentheses contain a mathematical expression, such as (2 + 2), but here they contain only one argument (x). Syntactically, they allow to avoid a space between the typeof operator and its argument, and some people like it.
Some people prefer typeof(x), although the typeof x syntax is much more common.
Summary
There are 8 basic data types in JavaScript.
Seven primitive data types:
number for numbers of any kind: integer or floating-point, integers are limited by ±(253-1).
bigint for integer numbers of arbitrary length.
string for strings. A string may have zero or more characters, there's no separate single-character type.
boolean for true/false.
null for unknown values – a standalone type that has a single value null.
undefined for unassigned values – a standalone type that has a single value undefined.
symbol for unique identifiers.
And one non-primitive data type:
object for more complex data structures.
The typeof operator allows us to see which type is stored in a variable.
Usually used as typeof x, but typeof(x) is also possible.
Returns a string with the name of the type, like "string".
For null returns "object" – this is an error in the language, it's not an object.
In the next chapters, we'll concentrate on primitive values and once we're familiar with them, we'll move on to objects.

Tasks
String quotes
What is the output of the script?

```
let name = "Ilya";

alert( `hello ${1}` ); // ?

alert( `hello ${"name"}` ); // ?

alert( `hello ${name}` ); // ?
```

3.6 Interaction: alert, prompt, confirm

As we'll be using the browser as our demo environment, let's see a couple of functions to interact with the user: alert, prompt and confirm.
alert

This one we've seen already. It shows a message and waits for the user to press "OK".
For example:
alert("Hello");
The mini-window with the message is called a modal window. The word "modal" means that the visitor can't interact with the rest of the page, press other buttons, etc, until they have dealt with the window. In this case – until they press "OK".
prompt
The function prompt accepts two arguments:
result = prompt(title, [default]);
It shows a modal window with a text message, an input field for the visitor, and the buttons OK/Cancel.
title
The text to show the visitor.
default
An optional second parameter, the initial value for the input field.
The square brackets in syntax [...]
The square brackets around default in the syntax above denote that the parameter is optional, not required.
The visitor can type something in the prompt input field and press OK. Then we get that text in the result. Or they can cancel the input by pressing Cancel or hitting the Esc key, then we get null as the result.
The call to prompt returns the text from the input field or null if the input was canceled.
For instance:
let age = prompt('How old are you?', 100);

alert(`You are ${age} years old!`); // You are 100 years old!
In IE: always supply a default
The second parameter is optional, but if we don't supply it, Internet Explorer will insert the text "undefined" into the prompt.
Run this code in Internet Explorer to see:
let test = prompt("Test");
So, for prompts to look good in IE, we recommend always providing the second argument:
let test = prompt("Test", "); // <-- for IE
confirm
The syntax:
result = confirm(question);
The function confirm shows a modal window with a question and two buttons: OK and Cancel.
The result is true if OK is pressed and false otherwise.
For example:
let isBoss = confirm("Are you the boss?");

alert(isBoss); // true if OK is pressed
Summary
We covered 3 browser-specific functions to interact with visitors:
alert
shows a message.
prompt
shows a message asking the user to input text. It returns the text or, if Cancel button or Esc is clicked, null.
confirm
shows a message and waits for the user to press “OK” or “Cancel”. It returns true for OK and false for Cancel/Esc.
All these methods are modal: they pause script execution and don’t allow the visitor to interact with the rest of the page until the window has been dismissed.
There are two limitations shared by all the methods above:
The exact location of the modal window is determined by the browser. Usually, it’s in the center.
The exact look of the window also depends on the browser. We can’t modify it.
That is the price for simplicity. There are other ways to show nicer windows and richer interaction with the visitor, but if “bells and whistles” do not matter much, these methods work just fine.

3.7 Basic operators, maths

We know many operators from school. They are things like addition +, multiplication *, subtraction -, and so on.
In this chapter, we’ll start with simple operators, then concentrate on JavaScript-specific aspects, not covered by school arithmetic.
Terms: “unary”, “binary”, “operand”
Before we move on, let’s grasp some common terminology.
An operand – is what operators are applied to. For instance, in the multiplication of 5 * 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.
An operator is *unary* if it has a single operand. For example, the unary negation - reverses the sign of a number:
let x = 1;

x = -x;
alert(x); // -1, unary negation was applied
An operator is *binary* if it has two operands. The same minus exists in binary form as well:
let x = 1, y = 3;
alert(y - x); // 2, binary minus subtracts values

Formally, in the examples above we have two different operators that share the same symbol: the negation operator, a unary operator that reverses the sign, and the subtraction operator, a binary operator that subtracts one number from another.

Maths

The following math operations are supported:

Addition +,
Subtraction -,
Multiplication *,
Division /,
Remainder %,
Exponentiation **.

The first four are straightforward, while % and ** need a few words about them.

Remainder %

The remainder operator %, despite its appearance, is not related to percents.

The result of a % b is the remainder of the integer division of a by b.

For instance:

```
alert( 5 % 2 ); // 1, the remainder of 5 divided by 2
alert( 8 % 3 ); // 2, the remainder of 8 divided by 3
alert( 8 % 4 ); // 0, the remainder of 8 divided by 4
```

Exponentiation **

The exponentiation operator a ** b raises a to the power of b.

In school maths, we write that as ab.

For instance:

```
alert( 2 ** 2 ); // 2² = 4
alert( 2 ** 3 ); // 2³ = 8
alert( 2 ** 4 ); // 2⁴ = 16
```

Just like in maths, the exponentiation operator is defined for non-integer numbers as well.

For example, a square root is an exponentiation by ½:

```
alert( 4 ** (1/2) ); // 2 (power of 1/2 is the same as a square root)
alert( 8 ** (1/3) ); // 2 (power of 1/3 is the same as a cubic root)
```

String concatenation with binary +

Let’s meet the features of JavaScript operators that are beyond school arithmetics.

Usually, the plus operator + sums numbers.

But, if the binary + is applied to strings, it merges (concatenates) them:

```
let s = "my" + "string";
alert(s); // mystring
```

Note that if any of the operands is a string, then the other one is converted to a string too.

For example:

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

See, it doesn't matter whether the first operand is a string or the second one.
Here's a more complex example:

```
alert(2 + 2 + '1' ); // "41" and not "221"
```

Here, operators work one after another. The first + sums two numbers, so it returns 4, then the next + adds the string 1 to it, so it's like 4 + '1' = '41'.

```
alert('1' + 2 + 2); // "122" and not "14"
```

Here, the first operand is a string, the compiler treats the other two operands as strings too. The 2 gets concatenated to '1', so it's like '1' + 2 = "12" and "12" + 2 = "122".
The binary + is the only operator that supports strings in such a way. Other arithmetic operators work only with numbers and always convert their operands to numbers.
Here's the demo for subtraction and division:

```
alert( 6 - '2' ); // 4, converts '2' to a number
alert( '6' / '2' ); // 3, converts both operands to numbers
```

Numeric conversion, unary +
The plus + exists in two forms: the binary form that we used above and the unary form.
The unary plus or, in other words, the plus operator + applied to a single value, doesn't do anything to numbers. But if the operand is not a number, the unary plus converts it into a number.
For example:

```
// No effect on numbers
let x = 1;
alert( +x ); // 1

let y = -2;
alert( +y ); // -2

// Converts non-numbers
alert( +true ); // 1
alert( +"" );   // 0
```

It actually does the same thing as Number(...), but is shorter.
The need to convert strings to numbers arises very often. For example, if we are getting values from HTML form fields, they are usually strings. What if we want to sum them?
The binary plus would add them as strings:

```
let apples = "2";
let oranges = "3";

alert( apples + oranges ); // "23", the binary plus concatenates strings
```

If we want to treat them as numbers, we need to convert and then sum them:

```
let apples = "2";
```

```
let oranges = "3";

// both values converted to numbers before the binary plus
alert( +apples + +oranges ); // 5

// the longer variant
// alert( Number(apples) + Number(oranges) ); // 5
```

From a mathematician's standpoint, the abundance of pluses may seem strange. But from a programmer's standpoint, there's nothing special: unary pluses are applied first, they convert strings to numbers, and then the binary plus sums them up.

Why are unary pluses applied to values before the binary ones? As we're going to see, that's because of their *higher precedence*.

Operator precedence

If an expression has more than one operator, the execution order is defined by their *precedence*, or, in other words, the default priority order of operators.

From school, we all know that the multiplication in the expression 1 + 2 * 2 should be calculated before the addition. That's exactly the precedence thing. The multiplication is said to have *a higher precedence* than the addition.

Parentheses override any precedence, so if we're not satisfied with the default order, we can use them to change it. For example, write (1 + 2) * 2.

There are many operators in JavaScript. Every operator has a corresponding precedence number. The one with the larger number executes first. If the precedence is the same, the execution order is from left to right.

Here's an extract from the precedence table (you don't need to remember this, but note that unary operators are higher than corresponding binary ones):

Precedence	**Name**	**Sign**
…	…	…
14	unary plus	+
14	unary negation	-
13	exponentiation	**
12	multiplication	*
12	Division	/
11	Addition	+
11	subtraction	-
…	…	…
2	assignment	=
…	…	…

As we can see, the “unary plus” has a priority of 14 which is higher than the 11 of “addition” (binary plus). That’s why, in the expression "+apples + +oranges", unary pluses work before the addition.

Assignment

Let’s note that an assignment = is also an operator. It is listed in the precedence table with the very low priority of 2.

That’s why, when we assign a variable, like x = 2 * 2 + 1, the calculations are done first and then the = is evaluated, storing the result in x.

```
let x = 2 * 2 + 1;

alert( x ); // 5
```

Assignment = returns a value

The fact of = being an operator, not a “magical” language construct has an interesting implication.

All operators in JavaScript return a value. That’s obvious for + and -, but also true for =.

The call x = value writes the value into x *and then returns it.*

Here’s a demo that uses an assignment as part of a more complex expression:

```
let a = 1;
let b = 2;

let c = 3 - (a = b + 1);

alert( a ); // 3
alert( c ); // 0
```

In the example above, the result of expression (a = b + 1) is the value that was assigned to a (that is 3). It is then used for further evaluations.

Funny code, isn’t it? We should understand how it works because sometimes we see it in JavaScript libraries.

Although, please don’t write the code like that. Such tricks definitely don’t make code clearer or readable.

Chaining assignments

Another interesting feature is the ability to chain assignments:

```
let a, b, c;

a = b = c = 2 + 2;

alert( a ); // 4
alert( b ); // 4
alert( c ); // 4
```

Chained assignments evaluate from right to left. First, the rightmost expression 2 + 2 is evaluated and then assigned to the variables on the left: c, b and a. At the end, all the variables share a single value.

Once again, for the purposes of readability it's better to split such code into few lines:

```
c = 2 + 2;
b = c;
a = c;
```

That's easier to read, especially when eye-scanning the code fast.

Modify-in-place

We often need to apply an operator to a variable and store the new result in that same variable.

For example:

```
let n = 2;
n = n + 5;
n = n * 2;
```

This notation can be shortened using the operators += and *=:

```
let n = 2;
n += 5; // now n = 7 (same as n = n + 5)
n *= 2; // now n = 14 (same as n = n * 2)

alert( n ); // 14
```

Short "modify-and-assign" operators exist for all arithmetical and bitwise operators: /=, -=, etc.

Such operators have the same precedence as a normal assignment, so they run after most other calculations:

```
let n = 2;

n *= 3 + 5; // right part evaluated first, same as n *= 8

alert( n ); // 16
```

Increment/decrement

Increasing or decreasing a number by one is among the most common numerical operations.

So, there are special operators for it:

Increment ++ increases a variable by 1:

```
let counter = 2;
counter++;          // works the same as counter = counter + 1, but is shorter
alert( counter ); // 3
```

Decrement -- decreases a variable by 1:

```
let counter = 2;
counter--;        // works the same as counter = counter - 1, but is shorter
alert( counter ); // 1
```

Important:

Increment/decrement can only be applied to variables. Trying to use it on a value like 5++ will give an error.

The operators ++ and -- can be placed either before or after a variable.

When the operator goes after the variable, it is in "postfix form": counter++.
The "prefix form" is when the operator goes before the variable: ++counter.
Both of these statements do the same thing: increase counter by 1.
Is there any difference? Yes, but we can only see it if we use the returned value of ++/--.
Let's clarify. As we know, all operators return a value. Increment/decrement is no exception. The prefix form returns the new value while the postfix form returns the old value (prior to increment/decrement).
To see the difference, here's an example:

```
let counter = 1;
let a = ++counter; // (*)

alert(a); // 2
```

In the line (*), the *prefix* form ++counter increments counter and returns the new value, 2. So, the alert shows 2.
Now, let's use the postfix form:

```
let counter = 1;
let a = counter++; // (*) changed ++counter to counter++

alert(a); // 1
```

In the line (*), the *postfix* form counter++ also increments counter but returns the *old* value (prior to increment). So, the alert shows 1.
To summarize:
If the result of increment/decrement is not used, there is no difference in which form to use:

```
let counter = 0;
counter++;
++counter;
alert( counter ); // 2, the lines above did the same
```

If we'd like to increase a value *and* immediately use the result of the operator, we need the prefix form:

```
let counter = 0;
alert( ++counter ); // 1
```

If we'd like to increment a value but use its previous value, we need the postfix form:

```
let counter = 0;
alert( counter++ ); // 0
```

Increment/decrement among other operators
The operators ++/-- can be used inside expressions as well. Their precedence is higher than most other arithmetical operations.
For instance:

```
let counter = 1;
alert( 2 * ++counter ); // 4
```

Compare with:
let counter = 1;
alert(2 * counter++); // 2, because counter++ returns the "old" value
Though technically okay, such notation usually makes code less readable. One line does multiple things – not good.
While reading code, a fast "vertical" eye-scan can easily miss something like counter++ and it won't be obvious that the variable increased.
We advise a style of "one line – one action":
let counter = 1;
alert(2 * counter);
counter++;
Bitwise operators
Bitwise operators treat arguments as 32-bit integer numbers and work on the level of their binary representation.
These operators are not JavaScript-specific. They are supported in most programming languages.
The list of operators:
AND (&)
OR (|)
XOR (^)
NOT (~)
LEFT SHIFT (<<)
RIGHT SHIFT (>>)
ZERO-FILL RIGHT SHIFT (>>>)
These operators are used very rarely, when we need to fiddle with numbers on the very lowest (bitwise) level. We won't need these operators any time soon, as web development has little use of them, but in some special areas, such as cryptography, they are useful. You can read the Bitwise Operators chapter on MDN when a need arises.
Comma
The comma operator is one of the rarest and most unusual operators. Sometimes, it's used to write shorter code, so we need to know it in order to understand what's going on.
The comma operator allows us to evaluate several expressions, dividing them with a comma , Each of them is evaluated but only the result of the last one is returned.
For example:
let a = (1 + 2, 3 + 4);

alert(a); // 7 (the result of 3 + 4)
Here, the first expression 1 + 2 is evaluated and its result is thrown away. Then, 3 + 4 is evaluated and returned as the result.
Comma has a very low precedence
Please note that the comma operator has very low precedence, lower than =, so parentheses are important in the example above.

Without them: a = 1 + 2, 3 + 4 evaluates + first, summing the numbers into a = 3, 7, then the assignment operator = assigns a = 3, and the rest is ignored. It's like (a = 1 + 2), 3 + 4.
Why do we need an operator that throws away everything except the last expression?
Sometimes, people use it in more complex constructs to put several actions in one line.
For example:

```
// three operations in one line
for (a = 1, b = 3, c = a * b; a < 10; a++) {
 ...
}
```

Such tricks are used in many JavaScript frameworks. That's why we're mentioning them. But usually they don't improve code readability so we should think well before using them.

Self-Assessment Exercise(s)

(1) What is the correct syntax for referring to an external script called "script.js"?
a) <script src="script.js"></script>
b) <script href="script.js"></script>
c) <script ref="script.js"></script>
d) <script name="script.js"></script>
Answer: a) <script src="script.js"></script>

(2) Which of the following is the correct way to declare a JavaScript variable?
a) variable x;
b) var x;
c) v x;
d) declare x;
Answer: b) var x;

(3) What will the following code output: console.log(typeof 42);?
a) "number"
b) "string"
c) "boolean"
d) "undefined"
Answer: a) "number"

(4) How do you create a function in JavaScript?
a) function:myFunction() {}
b) function myFunction() {}
c) create myFunction() {}
d) def myFunction() {}

Answer: b) function myFunction() {}
(5) Which event occurs when the user clicks on an HTML element?
a) onmouseover
b) onchange
c) onclick
d) onmouseclick
Answer: c) onclick

Conclusion

Mastering the basics of JavaScript is a fundamental step towards becoming a proficient web developer. This unit has equipped you with essential knowledge and skills, such as understanding variables, data types, operators, control structures, functions, and events. With these tools, you can create interactive and dynamic web pages that respond to user actions, enhancing the overall user experience. By learning to manipulate the DOM and implement best practices for clean and efficient code, you are now prepared to tackle more advanced JavaScript concepts and projects. Continue practicing and experimenting with JavaScript to solidify your understanding and expand your capabilities in web development.

4.0 Summary

The Basics of JavaScript unit provides a comprehensive introduction to the core concepts and syntax of JavaScript, a crucial programming language for creating interactive and dynamic web content. The unit covers fundamental topics including variables, data types, operators, control structures, functions, and events, all of which are essential for building responsive web applications. This foundational knowledge equips you to write basic JavaScript programs, integrate them into web pages, and follow best practices for clean, efficient code, laying the groundwork for more advanced studies in web development.

5.0 References/Further Readings

Zakas, N. C. (2011). Professional JavaScript for web developers. John Wiley & Sons.

Hartl, M. (2022). Learn Enough JavaScript to Be Dangerous: A Tutorial Introduction to Programming with JavaScript.

Chen, E., & Asta, M. (2022). Using Jupyter tools to design an interactive textbook to guide undergraduate research in materials informatics.

Abbas, S. H., Siddiqui, N., & Ali, S. (2021). Internet And Web Programming+ Projects.

Unit 2 JavaScript Functions

Contents

1.0 Introduction

JavaScript functions are fundamental building blocks in programming that enable developers to encapsulate code for reuse, modularity, and better organization. A function in JavaScript is a set of instructions that performs a specific task or calculates a value, which can be invoked as needed throughout the code. By defining a function once and reusing it multiple times, developers can write cleaner, more efficient, and more maintainable code. Functions not only help in reducing redundancy but also make the code easier to read and debug. Understanding how to create and utilize functions is essential for anyone looking to master JavaScript and build dynamic, interactive web applications. In this unit, we will delve into the various aspects of JavaScript functions, starting with the basics of function declaration and invocation. We will explore different types of functions, including named functions, anonymous functions, and arrow functions, each with its own unique syntax and use cases.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss JavaScript functions
- explain function Declaration
- discussthe evaluation of default parameters
- explain naming a function

3.0 Main Content

3.1 Functions in JavaScript

JavaScript functions are essential for organizing code and executing specific tasks. They contain sets of instructions that run when triggered by events or actions. In this article, we'll explore the syntax, parameters, return values, and execution contexts of JavaScript functions. Through practical examples, we'll provide a clear understanding of how to use functions effectively in web development.
A JavaScript function is executed when "something" invokes it (calls it).
A basic JavaScript function, here we create a function that divides the 1st element by the second element.

```
function myFunction(g1, g2) {
   return g1 / g2;
}
const value = myFunction(8, 2); // Calling the function
console.log(value);
```

You must already have seen some commonly used functions in JavaScript like alert(), which is a built-in function in JavaScript. But JavaScript allows us to create user-defined functions also. We can create functions in JavaScript using the keyword `function`.
Quite often we need to perform a similar action in many places of the script.
For example, we need to show a nice-looking message when a visitor logs in, logs out and maybe somewhere else.
Functions are the main "building blocks" of the program. They allow the code to be called many times without repetition.
We've already seen examples of built-in functions, like alert(message), prompt(message, default) and confirm(question). But we can create functions of our own as well.

3.2 Function Declaration

To create a function we can use a *function declaration*.
It looks like this:

```
function showMessage() {
alert( 'Hello everyone!' );
}
```

The function keyword goes first, then goes the *name of the function*, then a list of *parameters* between the parentheses (comma-separated, empty in the example above, we'll see examples later) and finally the code of the function, also named "the function body", between curly braces.

```
function name(parameter1, parameter2, ... parameterN) {
 // body
}
```

Our new function can be called by its name: showMessage().
For instance:

```
function showMessage() {
alert( 'Hello everyone!' );
}

showMessage();
showMessage();
```

The call showMessage() executes the code of the function. Here we will see the message two times.
This example demonstrates one of the main purposes of functions: to avoid code duplication.
If we ever need to change the message or the way it is shown, it's enough to modify the code in one place: the function which outputs it.

Local variables

A variable declared inside a function is only visible inside that function.
For example:

```
function showMessage() {
  let message = "Hello, I'm JavaScript!"; // local variable

alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Error! The variable is local to the function
```

Outer variables

A function can access an outer variable as well, for example:

```
let userName = 'John';

function showMessage() {
  let message = 'Hello, ' + userName;
  alert(message);
}

showMessage(); // Hello, John
```

The function has full access to the outer variable. It can modify it as well.
For instance:

```
let userName = 'John';

function showMessage() {
```

```
  userName = "Bob"; // (1) changed the outer variable

  let message = 'Hello, ' + userName;
  alert(message);
}

alert( userName ); // John before the function call

showMessage();

alert( userName ); // Bob, the value was modified by the function
```

The outer variable is only used if there's no local one.

If a same-named variable is declared inside the function then it *shadows* the outer one. For instance, in the code below the function uses the local userName. The outer one is ignored:

```
let userName = 'John';

function showMessage() {
  let userName = "Bob"; // declare a local variable

  let message = 'Hello, ' + userName; // Bob
  alert(message);
}

// the function will create and use its own userName
showMessage();

alert( userName ); // John, unchanged, the function did not access the outer variable
```

Global variables

Variables declared outside of any function, such as the outer userName in the code above, are called *global*.

Global variables are visible from any function (unless shadowed by locals).

It's a good practice to minimize the use of global variables. Modern code has few or no globals. Most variables reside in their functions. Sometimes though, they can be useful to store project-level data.

Parameters

We can pass arbitrary data to functions using parameters.

In the example below, the function has two parameters: from and text.

```
function showMessage(from, text) { // parameters: from, text
alert(from + ': ' + text);
}

showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
```

showMessage('Ann', "What's up?"); // Ann: What's up? (**)
When the function is called in lines (*) and (**), the given values are copied to local variables from and text. Then the function uses them.
Here's one more example: we have a variable from and pass it to the function. Please note: the function changes from, but the change is not seen outside, because a function always gets a copy of the value:

```
function showMessage(from, text) {

  from = '*' + from + '*'; // make "from" look nicer

alert( from + ': ' + text );
}

let from = "Ann";

showMessage(from, "Hello"); // *Ann*: Hello

// the value of "from" is the same, the function modified a local copy
alert( from ); // Ann
```

When a value is passed as a function parameter, it's also called an *argument*.
In other words, to put these terms straight:
A parameter is the variable listed inside the parentheses in the function declaration (it's a declaration time term).
An argument is the value that is passed to the function when it is called (it's a call time term).
We declare functions listing their parameters, then call them passing arguments.
In the example above, one might say: "the function showMessage is declared with two parameters, then called with two arguments: from and "Hello"".

Default values
If a function is called, but an argument is not provided, then the corresponding value becomes undefined.
For instance, the aforementioned function showMessage(from, text) can be called with a single argument:

```
showMessage("Ann");
```

That's not an error. Such a call would output "*Ann*: undefined". As the value for text isn't passed, it becomes undefined.
We can specify the so-called "default" (to use if omitted) value for a parameter in the function declaration, using =:

```
function showMessage(from, text = "no text given") {
alert( from + ": " + text );
}
```

showMessage("Ann"); // Ann: no text given
Now if the text parameter is not passed, it will get the value "no text given".
The default value also jumps in if the parameter exists, but strictly equals undefined, like this:
showMessage("Ann", undefined); // Ann: no text given
Here "no text given" is a string, but it can be a more complex expression, which is only evaluated and assigned if the parameter is missing. So, this is also possible:

```
function showMessage(from, text = anotherFunction()) {
  // anotherFunction() only executed if no text given
  // its result becomes the value of text
}
```

3.3 Evaluation of default parameters

In JavaScript, a default parameter is evaluated every time the function is called without the respective parameter.
In the example above, anotherFunction() isn't called at all, if the text parameter is provided.
On the other hand, it's independently called every time when text is missing.
Default parameters in old JavaScript code
Several years ago, JavaScript didn't support the syntax for default parameters. So people used other ways to specify them.
Nowadays, we can come across them in old scripts.
For example, an explicit check for undefined:

```
function showMessage(from, text) {
  if (text === undefined) {
    text = 'no text given';
  }

alert( from + ": " + text );
}
```

…Or using the || operator:

```
function showMessage(from, text) {
  // If the value of text is falsy, assign the default value
  // this assumes that text == "" is the same as no text at all
  text = text || 'no text given';
  ...
}
```

Alternative default parameters
Sometimes it makes sense to assign default values for parameters at a later stage after the function declaration.
We can check if the parameter is passed during the function execution, by comparing it with undefined:

```
function showMessage(text) {
  // ...

  if (text === undefined) { // if the parameter is missing
    text = 'empty message';
  }

  alert(text);
}

showMessage(); // empty message
```

…Or we could use the || operator:

```
function showMessage(text) {
  // if text is undefined or otherwise falsy, set it to 'empty'
  text = text || 'empty';
  ...
}
```

Modern JavaScript engines support the nullish coalescing operator ??, it's better when most falsy values, such as 0, should be considered "normal":

```
function showCount(count) {
  // if count is undefined or null, show "unknown"
alert(count ?? "unknown");
}

showCount(0); // 0
showCount(null); // unknown
showCount(); // unknown
```

Returning a value

A function can return a value back into the calling code as the result.

The simplest example would be a function that sums two values:

```
function sum(a, b) {
  return a + b;
}

let result = sum(1, 2);
alert( result ); // 3
```

The directive return can be in any place of the function. When the execution reaches it, the function stops, and the value is returned to the calling code (assigned to result above).

There may be many occurrences of return in a single function. For instance:

```
function checkAge(age) {
  if (age >= 18) {
    return true;
  } else {
```

```
    return confirm('Do you have permission from your parents?');
  }
}

let age = prompt('How old are you?', 18);

if ( checkAge(age) ) {
alert( 'Access granted' );
} else {
alert( 'Access denied' );
}
```

It is possible to use return without a value. That causes the function to exit immediately.

For example:

```
function showMovie(age) {
  if ( !checkAge(age) ) {
    return;
  }

alert( "Showing you the movie" ); // (*)
  // ...
}
```

In the code above, if checkAge(age) returns false, then showMovie won't proceed to the alert.

A function with an empty return or without it returns undefined

If a function does not return a value, it is the same as if it returns undefined:

```
function doNothing() { /* empty */ }

alert( doNothing() === undefined ); // true
```

An empty return is also the same as return undefined:

```
function doNothing() {
  return;
}

alert( doNothing() === undefined ); // true
```

Never add a newline between return and the value

For a long expression in return, it might be tempting to put it on a separate line, like this:

```
return
 (some + long + expression + or + whatever * f(a) + f(b))
```

That doesn't work, because JavaScript assumes a semicolon after return. That'll work the same as:

```
return;
 (some + long + expression + or + whatever * f(a) + f(b))
```

So, it effectively becomes an empty return.

If we want the returned expression to wrap across multiple lines, we should start it at the same line as return. Or at least put the opening parentheses there as follows:

```
return (
  some + long + expression
  + or +
  whatever * f(a) + f(b)
  )
```

And it will work just as we expect it to.

3.4 Naming a function

Functions are actions. So their name is usually a verb. It should be brief, as accurate as possible and describe what the function does, so that someone reading the code gets an indication of what the function does.

It is a widespread practice to start a function with a verbal prefix which vaguely describes the action. There must be an agreement within the team on the meaning of the prefixes.

For instance, functions that start with "show" usually show something.

Function starting with…

"get…" – return a value,

"calc…" – calculate something,

"create…" – create something,

"check…" – check something and return a boolean, etc.

Examples of such names:

```
showMessage(..)     // shows a message
getAge(..)          // returns the age (gets it somehow)
calcSum(..)         // calculates a sum and returns the result
createForm(..)      // creates a form (and usually returns it)
checkPermission(..) // checks a permission, returns true/false
```

With prefixes in place, a glance at a function name gives an understanding what kind of work it does and what kind of value it returns.

One function – one action

A function should do exactly what is suggested by its name, no more.

Two independent actions usually deserve two functions, even if they are usually called together (in that case we can make a 3rd function that calls those two).

A few examples of breaking this rule:

getAge – would be bad if it shows an alert with the age (should only get).

createForm – would be bad if it modifies the document, adding a form to it (should only create it and return).

checkPermission – would be bad if it displays the access granted/denied message (should only perform the check and return the result).

These examples assume common meanings of prefixes. You and your team are free to agree on other meanings, but usually they're not much different. In any case, you should have a firm understanding of what a prefix means, what a prefixed function can and cannot do. All same-prefixed functions should obey the rules. And the team should share the knowledge.

Ultrashort function names

Functions that are used *very often* sometimes have ultrashort names.

For example, the jQuery framework defines a function with $. The Lodash library has its core function named _.

These are exceptions. Generally function names should be concise and descriptive.

Functions == Comments

Functions should be short and do exactly one thing. If that thing is big, maybe it's worth it to split the function into a few smaller functions. Sometimes following this rule may not be that easy, but it's definitely a good thing.

A separate function is not only easier to test and debug – its very existence is a great comment!

For instance, compare the two functions showPrimes(n) below. Each one outputs prime numbers up to n.

The first variant uses a label:

```
function showPrimes(n) {
  nextPrime: for (let i = 2; i < n; i++) {

    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

alert( i ); // a prime
  }
}
```

The second variant uses an additional function isPrime(n) to test for primality:

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);  // a prime
  }
}
```

```
function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if ( n % i == 0) return false;
  }
  return true;
}
```

The second variant is easier to understand, isn't it? Instead of the code piece, we see the name of the action (isPrime). Sometimes people refer to such code as *self-describing*.

So, functions can be created even if we don't intend to reuse them. They structure the code and make it readable.

Self-Assessment Exercise(s)

(1) What is the correct way to define a function in JavaScript?
A) function myFunction = {}
B) function myFunction() {}
C) def myFunction() {}
D) function: myFunction() {}
Answer: B) function myFunction() {}

(2) Which of the following is a characteristic of an arrow function in JavaScript?
A) It must have a name.
B) It cannot be assigned to a variable.
C) It inherits this value from the enclosing scope.
D) It requires the function keyword.
Answer: C) It inherits this value from the enclosing scope.

(3) What will be the output of the following code snippet?
javascript
Copy code

```
function greet() {
return "Hello, World!";
}
console.log(greet());
```

A) Hello
B) World
C) Hello, World!
D) undefined
Answer: C) Hello, World!

(4) Which of the following statements is true about closures in JavaScript?
A) Closures are used to define global variables.

B) A closure is a function that has access to its scope, the scope of the outer function, and the global scope.
C) Closures cannot access variables outside of their function scope.
D) Closures are only used with arrow functions.

Answer: B) A closure is a function that has access to its scope, the scope of the outer function, and the global scope.
(5) What is a higher-order function in JavaScript?
A) A function that can only be called once.
B) A function that returns another function or takes one or more functions as arguments.
C) A function that executes immediately after its definition.
D) A function that cannot be assigned to a variable.

Answer: B) A function that returns another function or takes one or more functions as arguments.

Conclusion

Mastering JavaScript functions is crucial for developing robust and efficient web applications. Functions enable you to write modular, reusable, and maintainable code, which is essential for handling complex programming tasks. By understanding different types of functions, such as name, anonymous, and arrow functions, and key concepts like scope, closures, and higher-order functions, you can leverage the full power of JavaScript to create dynamic and interactive user experiences. As you continue to practice and apply these concepts, you'll become more proficient in writing clean and effective code, significantly enhancing your programming skills and capabilities.

4.0 Summary

A function declaration looks like this:

```
function name(parameters, delimited, by, comma) {
  /* code */
}
```

Values passed to a function as parameters are copied to its local variables.
A function may access outer variables. But it works only from the inside out. The code outside of the function doesn’t see its local variables.
A function can return a value. If it doesn’t, then its result is undefined.
To make the code clean and easy to understand, it’s recommended to use mainly local variables and parameters in the function, not outer variables.

It is always easier to understand a function that gets parameters, works with them, and returns a result than a function that gets no parameters, but modifies outer variables as a side effect.
Function naming:
A name should clearly describe what the function does. When we see a function call in the code, a good name instantly gives us an understanding of what it does and returns.

A function is an action, so function names are usually verbal.
There exist many well-known function prefixes like create…, show…, get…, check…, and so on. Use them to hint at what a function does.

Functions are the main building blocks of scripts. Now we've covered the basics, so we actually can start creating and using them. But that's only the beginning of the path. We are going to return to them many times, going more deeply into their advanced features.

5.0 References/Further Reading

Zakas, N. C. (2011). Professional JavaScript for web developers. John Wiley & Sons.

Hartl, M. (2022). Learn Enough JavaScript to Be Dangerous: A Tutorial Introduction to Programming with JavaScript.

Chen, E., & Asta, M. (2022). Using Jupyter tools to design an interactive textbook to guide undergraduate research in materials informatics.

Abbas, S. H., Siddiqui, N., & Ali, S. (2021). Internet And Web Programming+ Projects.

Unit 3 Document Object Model (DOM) Manipulation

Contents

1.0 Introduction

The Document Object Model (DOM) is a crucial concept in web development, serving as the bridge between web pages and the programming languages that manipulate them. At its core, the DOM is a programming interface for HTML and XML documents. It represents the structure of a document as a tree of nodes, where each node corresponds to a part of the document, such as an element, attribute, or piece of text. This hierarchical model allows developers to programmatically access and modify the content, structure, and styling of web pages, making dynamic and interactive user experiences possible. DOM manipulation involves using scripting languages like JavaScript to interact with the DOM. By targeting specific nodes within the DOM, developers can change how a webpage appears and behaves without needing to reload the page. This capability is fundamental to creating responsive and interactive web applications. For instance, DOM manipulation can be used to update the content of a page based on user input, animate elements, handle events, and more. Understanding how to efficiently navigate and alter the DOM is essential for modern web development, enabling developers to build sophisticated, user-friendly interfaces.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- discuss the foundation of DOM manipulation
- explain select elements in the DOM
- discuss creating a new element
- explain DOM events

3.0 Main Content

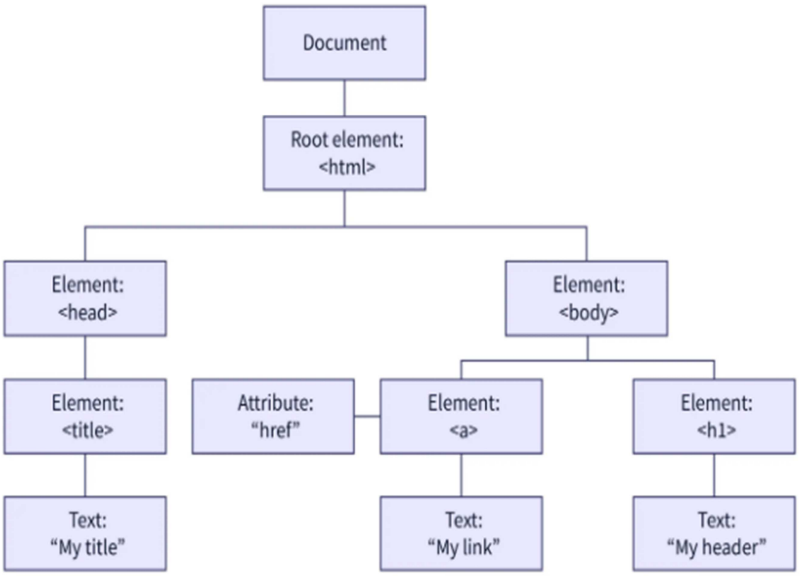
3.1 Foundation of DOM manipulation

DOM manipulation in JavaScript is an important factor while creating a web application using HTML and JavaScript. It is the process of interacting with the DOM API to change or modify an HTML document that will be displayed in a web browser. This HTML document can be changed to add or remove elements, update existing elements, rearrange existing elements, etc.

By manipulating the DOM, we can create web applications that update the data in a web page without refreshing the page and can change its layout without doing a refresh. Throughout the document, items can be deleted, moved, or rearranged.

Definition of the DOM and Basic concepts
The DOM in DOM manipulation in javascript stands for **Document Object Model**. The Document Object Model (DOM) is a tree-like structure illustrating the hierarchical relationship between various HTML elements. It can be easily explained as a tree of nodes generated by the browser. Each node has unique properties and methods that can be changed using JavaScript.

A visual representation of the DOM tree is shown in the image below.



The document is the core/foundation of the DOM.
HTML root element is the child of the document object.
Body and **Head** elements are the children of the HTML element and siblings to each other.
The title element is the parent to the text node: "my text" and the child of the head element.
a tag and **h1** tag are the children of the body element and siblings to each other.
href attribute is the children of the **a**(anchor) tag.
The DOM is referred to as a programming API for XML and HTML documents. DOM defines the logical structure of documents and the methods for accessing and changing them are specified by the DOM.
A few Points to Keep in Mind

- A node can have more than one child.
- Siblings are nodes with the same parent like brothers or sisters.
- Except for the top node, which has no parent, each node has exactly one parent.

3.2 How to Select Elements in the DOM

To change or modify an element in the DOM, you need to select that specific element. Thus, JavaScript has six methods to select an element from a document in dom manipulation in javascript.

- **getElementById:** returns an element whose id matches a passed string. Since the ids of elements are unique, this is the fastest way to select an element.
- **getElementsByTagName:** returns a collection of all the elements present in the document that have the specified tag name, in the order of their appearance in the document.

- **getElementsByClassName:** returns an HTMLCollection of elements that match the passed class name. Bypassing the class names separated by whitespace, we can search for multiple class names.
- **getElementsByName:** returns a NodeList Collection of the elements that match the value of the name attribute with the passed string.
- **querySelector:** returns the very first element within the document that matches the given selector. It only returns the element that matches with one of the specified CSS selectors, or a group of selectors.
- **querySelectorAll:** returns a static NodeList of elements that matches with one or a group of selectors. If no element matches, an empty NodeList is returned.

How to Traverse and Move around the DOM

With the HTML DOM, we can navigate through the tree nodes and access them using node relationships that we have discussed earlier parent, children, siblings, etc.

In this section, we will learn how to get the parent element, and children of an element, and siblings of an element.

Get the Parent Element

To get the parent node of a particular node in the DOM tree, we can use the **parentNode** property. for example,

let parent = node.parentNode;

ParentNode is a read-only object. There is no parent for the Document and DocumentFragment nodes. As a result, the parentNode is always empty. The parentNode of a newly created node that hasn't been connected to the DOM tree will also be null.

Get Child Elements

To Get all child elements use the **firstChild** property that returns the first child element of a specified element.

let firstChild = parentElement.firstChild;

To Get the last child element use the **lastChild** property that returns the first element node, text node, or comment node.

let lastChild = parentElement.lastChild;

To Get all child elements use the **childNodes** property that returns a live NodeList of child elements of a specified element.

let children = parentElement.childNodes;

Get Siblings of an Element

To Get the next siblings use the **nextElementSibling** property

let nextSibling = currentNode.nextElementSibling;

To Get the previous siblings use

the **previousElementSibling** property.

let current = document.querySelector('.current');
let prevSibling = currentNode.previousElementSibling;
How to Manipulate Elements in the DOM

3.3 Create a New Element

The document.createElement() returns a new Node with the Element type. It takes an HTML tag name as a parameter.
let div = document.createElement('div');
Get and Set the Text Content of a Node
The textContent property returns the concatenation of the textContent of all child nodes and does not include the comments.
let text = node.textContent;
Get and Set the HTML Content of an Element
The innerHTML is a property of the Element that allows us to get or set the HTML markup contained within the element.
element.innerHTML = 'new content';
Append a Node to a List of Child Nodes of a Particular Parent Node
The appendChild() method allows us to insert a node at the end of the list of child nodes of a particular parent node.
parentNode.appendChild(childNode);
Insert One Element Before an Existing Node as a Child Node of a Specified Parent Node
The insertBefore() JavaScript method takes two parameters, the newNode, and the existingNode. insertBefore() returns the inserted child node.
parentNode.insertBefore(newNode, existingNode);
Replace a Child Element by a New Element
The replaceChild() JavaScript method takes two parameters to replace our first element with the newly created one.
parentNode.replaceChild(newChild, oldChild);
Remove Child Elements of a Node
The removeChild() JavaScript method takes just one parameter i.e the element you want to remove.
let childNode = parentNode.removeChild(childNode);
Clone an Element and All of Its Descendants
This method allows us to clone an element. The cloneNode() method takes an optional parameter deep. If the deep is true, then the original node and all of its descendants are already cloned, false otherwise.
let clonedNode = originalNode.cloneNode(deep);

3.4 Working with Events

Here are some events used in dom manipulation in javascript:

- **Handling events:** includes HTML event handler attribute, element's event handler property, and addEventListener().

- **Page Load Events:** includes DOMContentLoaded, load, beforeunload, and unload.
- **load event:** includes dependent resources like JavaScript files, CSS files, and images.
- **Unload event:** The unload event is fired after before unload event and pagehide event.
- **Mouse events:** includes mousedown, mouseup, and click.
- **Keyboard events:** includes keydown, keypress, and keyup.

Scroll events: includes scrollX and scrollY properties that returns the number of pixels that the document is currently scrolled horizontally and vertically.

Manipulating Element's Styles
style property
The style property returns the read-only CSSStyleDeclaration object that includes a list of CSS properties.
element.style.color = 'red';
getComputedStyle()
The getComputedStyle() method returns the object that contains the computed style of an element.
element.style.color = 'red';
className Property
The className property returns a space-separated list of CSS classes of the element as a string.
element.className
classList Property
The classList returns a live collection of CSS classes. It is a read-only property of an element.
const classes = element.classList;
Event Handling in the DOM
Since the beginning of the language, event handling has been a part of dom manipulation in javascript. They correspond to specific, user-imitated actions within the webpage, such as the moving of your mouse over a link, clicking on a link, or submitting a form. Thanks to event handling, our scripts are more interactive and are able to perform certain actions depending on the users.
The DOM of modern web browsers such as NS6+, IE5+, and Firefox provide expanded methods and flexibility for capturing events.

3.5 Scripting Web Forms

JavaScript Form
To create a form in HTML, you use the <form> element, for example
<!DOCTYPE html>
<html lang="en">

```
<head>
<title>JavaScript Form Demo</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="css/style.css" />
</head>
<body>
<div class="container">
<form action="signup.html" method="post" id="signup">
<h1>Sign Up</h1>
<div class="field">
<label for="name">Name:</label>
<input type="text" id="name" name="name" placeholder="Enter your fullname" />
<small></small>
</div>
<div class="field">
<label for="email">Email:</label>
<input type="text" id="email" name="email" placeholder="Enter your email address" />
<small></small>
</div>
<div class="field">
<button type="submit" class="full">Subscribe</button>
</div>
</form>
</div>
<script>
        // show a message with a type of the input
   function showMessage(input, message, type) {
   // get the small element and set the message
   const msg = input.parentNode.querySelector("small");
msg.innerText = message;
   // update the class for the input
input.className = type ? "success": "error";
   return type;
   }

   function showError(input, message) {
   return showMessage(input, message, false);
   }
   function showSuccess(input) {
   return showMessage(input, "", true);
   }
   function hasValue(input, message) {
   if (input.value.trim() === "") {
```

```
      return showError(input, message);
   }
   return showSuccess(input);
   }

   function validateEmail(input, requiredMsg, invalidMsg) {
   // check if the value is not empty
   if (!hasValue(input, requiredMsg)) {
      return false;
   }
   // validate email format
   const emailRegex = (".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}\.[0-9]{1,3}\])

   const email = input.value.trim();
   if (!emailRegex.test(email)) {
      return showError(input, invalidMsg);
   }
   return true;
   }

   const form = document.querySelector("#signup");

   const NAME_REQUIRED = "Please enter your name";
   const EMAIL_REQUIRED = "Please enter your email";
   const EMAIL_INVALID = "Please enter a correct email address
format";

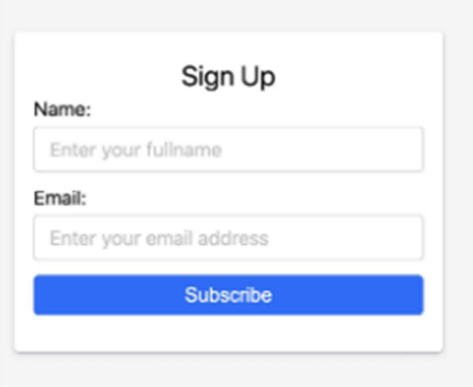
form.addEventListener("submit", function (event) {
   // stop form submission
event.preventDefault();

   // validate the form
   let nameValid = hasValue(form.elements["name"],
NAME_REQUIRED);
   let emailValid = validateEmail(form.elements["email"],
EMAIL_REQUIRED, EMAIL_INVALID);
   // if valid, submit the form.
   if (nameValid && emailValid) {
alert("Demo only. No form was posted.");
   }
   });


</script>
</body>
```

```
</html>
```

Output



Radio Button

To know which radio button is checked, we need to use the value attribute, For example

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>JavaScript Radio Button</title>
</head>
<body>
<p>Select your size:</p>
<div>
<input type="radio" name="size" value="XS" id="xs">
<label for="xs">XS</label>
</div>
<div>
<input type="radio" name="size" value="S" id="s">
<label for="s">S</label>
</div>
<div>
<input type="radio" name="size" value="M" id="m">
<label for="m">M</label>
</div>
<div>
<input type="radio" name="size" value="L" id="l">
<label for="l">L</label>
</div>
<div>
<input type="radio" name="size" value="XL" id="xl">
<label for="xl">XL</label>
```

```
</div>
<div>
<input type="radio" name="size" value="XXL" id="xxl">
<label for="xxl">XXL</label>
</div>
<p>
<button id="btn">Show Selected Value</button>
</p>

<p id="output"></p>

<script>
      const btn = document.querySelector('#btn');
      const radioButtons =
document.querySelectorAll('input[name="size"]');
btn.addEventListener("click", () => {
         let selectedSize;
         for (const radioButton of radioButtons) {
            if (radioButton.checked) {
               selectedSize = radioButton.value;
               break;
            }
         }
         // show the output:
output.innerText = selectedSize ? `You selected ${selectedSize}` :
`You haven't selected any size`;
      });
</script>
</body>
</html>
```

Output

Checkbox

To create a checkbox, you use the <input> element with the type of checkbox, for example

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>JavaScript Checkbox</title>
</head>

<body>
<label for="accept">
<input type="checkbox" id="accept" name="accept"> Accept
</label>

<button id="btn">Submit</button>
<script>
      const cb = document.querySelector('#accept');
      const btn = document.querySelector('#btn');
btn.onclick = () => {
        alert(cb.value);
      };
</script>
</body>

</html>
```

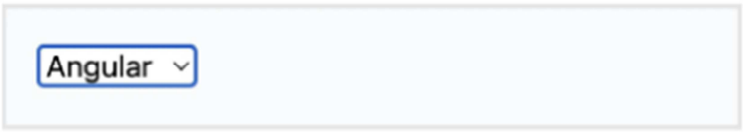
Output

Select Box

To create a <select> element, you use the <select> and <option> elements. For example:

```
<select id="framework">
<option value="1">Angular</option>
<option value="2">React</option>
<option value="3">Vue.js</option>
<option value="4">Ember.js</option>
</select>
```

Output



Handling Input Event
The input event fires each time whenever the value of the <input>, <select>, or <textarea> element updates.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>JavaScript input Event Demo</title>
</head>
<body>
<label for="message">Message</label>
<input placeholder="Enter some text" id="message" name="message">
<p id="result"></p>
<script>
    const message = document.querySelector('#message');
    const result = document.querySelector('#result');
message.addEventListener('input', function () {
result.textContent = this.value;
    });
</script>
</body>
</html>
```

Output

Web developers can perform DOM manipulation using JavaScript. This includes dynamically changing page content, adding new elements, removing existing elements, and updating style properties.
Changing Element Content:

We can use innerHTML or textContent properties to change the text content of an element within the DOM.

```
var element = document.getElementById("myElementId");
element.innerHTML = "New content"; // or element.textContent = "New content";
```

3.6 DOM Events

Interactions on web pages are carried out using events to enable users to interact with the page. Events represent interactions such as a user clicking a mouse, pressing a key, submitting a form, etc. Using JavaScript, we can listen for these events and react to user interactions.

Event Listeners:

We can use the addEventListener() function to listen for a specific event. For example, to listen for a button click event:

```
var button = document.getElementById("myButton");
button.addEventListener("click", function() {
  // Actions to be taken when a click event occurs
});
```

Common Events:

click: For click on an item.

mouseover and mouseout: For hover over and move away from an element.

keydown and keyup: For pressing a key and for it to work when you unclick it.

submit: For submit a form.

Event Object:

Event listeners receive an event object containing information about the event when it occurs. This object contains information such as the type of the event, its target, and key information.

Thanks to event listeners, you can make your web page responsive to user interactions and improve the user experience.

Asynchronous DOM Operations

In modern web applications, it is common to retrieve data from the server and update it in the DOM to dynamically update pages. Such operations must be performed asynchronously. Asynchronous operations allow the page to continue without being blocked by other operations, so the page remains responsive and user-friendly during user interactions.

Asynchronous Data Retrieval with Fetch API:

The Fetch API is a new asynchronous data retrieval method used in modern browsers. It attracts attention with its easier use and promise-based structure.

```
fetch("data.json")
  .then(response => response.json())
  .then(data => {
    // Using data within the DOM
  })
  .catch(error => {
    console.error("Error: ", error);
  });
```

Asynchronous DOM Update:
It is possible to update the DOM content using data received asynchronously. For example, we can populate a list asynchronously.

```
fetch("data.json")
  .then(response => response.json())
  .then(data => {
    var list = document.getElementById("myList");
    data.forEach(item => {
      var listItem = document.createElement("li");
      listItem.textContent = item.text;
      list.appendChild(listItem);
    });
  })
  .catch(error => {
    console.error("Error: ", error);
  });
```

3.7 Security and DOM

Security is an important factor when performing DOM manipulation. Malicious users may attempt to manipulate web pages for malicious purposes. Therefore, you must take some precautions to ensure safety.
XSS (Cross-Site Scripting) Attacks:
XSS attacks occur when malicious users inject JavaScript code on the page. In this case, it is important to process user data securely and expose external data securely.

Adding Data via Secure Ways:
Safe methods such as textContent or createElement should be used when adding user input into the DOM. You should not include user data directly with innerHTML containing it.
Using CSP (Content Security Policy):
CSP is used to control where the page's resources can be loaded from. CSP can help prevent XSS attacks and data leaks. CSP policies can be specified in the page title or in HTTP responses.
Event Listeners with Secure Methods:

When listening to user interactions, addEventListener should be used instead of features like onclick or onmouseover. This helps manage interactions on the DOM more securely.
DOM is a fundamental concept in the world of modern web development. DOM, a tool used to make pages dynamic and interactive, is the primary way to interact with web browsers. DOM represents HTML and XML documents as a tree structure, which treats each element in web pages as an object. Changes made to these objects make it possible to update the content of the web page, add new elements, remove existing ones, and listen to interactions. Using the DOM correctly and securely is the key to developing powerful and effective web applications.

Self-Assessment Exercise(s)

(1) What does the Document Object Model (DOM) represent in a web document?
A) The style of the document
B) The structure of the document as a tree of nodes
C) The server-side scripts of the document
D) The user interactions with the document
Answer: B) The structure of the document as a tree of nodes
(2) Which of the following methods is used to select an element by its ID in the DOM?
A) document.getElementsByClassName
B) document.getElementsByTagName
C) document.getElementById
D) document.querySelectorAll
Answer: C) document.getElementById
(3) What is the correct way to change the text content of an HTML element with the id "header" to "Welcome"?
A) document.getElementById("header").innerHTML = "Welcome";
B) document.getElementById("header").value = "Welcome";
C) document.getElementById("header").text = "Welcome";
D) document.getElementById("header").innerText = "Welcome";
Answer: D) document.getElementById("header").innerText = "Welcome";
(4) Which DOM method is used to create a new HTML element?
A) document.createElement()
B) document.newElement()
C) document.appendChild()
D) document.createNode()
Answer: A) document.createElement()
(5) How can you add a new class to an existing HTML element using JavaScript?

A) element.setAttribute("class", "new-class");
B) element.className += " new-class";
C) element.classList.add("new-class");
D) element.addClass("new-class");
Answer: C) element.classList.add("new-class");

Conclusion
Mastering Document Object Model (DOM) manipulation empowers developers to create dynamic and interactive web experiences. By understanding how to access, modify, and update elements within the DOM tree, developers can breathe life into static web pages, enabling them to respond dynamically to user input and events. Through techniques such as selecting elements, changing their attributes, and manipulating their content, developers can craft seamless user interfaces and enhance the overall user experience. However, it's crucial to maintain code readability, performance, and accessibility while implementing DOM manipulation to ensure scalability and compatibility across various browsers and devices. Embracing best practices and staying updated with emerging technologies will further solidify developers' mastery of DOM manipulation, enabling them to build robust and engaging web applications for diverse audiences.

4.0 Summary

The Document Object Model (DOM) Manipulation unit focuses on the essential concept of interacting with web documents through JavaScript. It delves into how web browsers construct a tree-like representation of HTML documents, where each element, attribute, and text node is a part of the hierarchy. Through DOM manipulation, developers gain the ability to dynamically modify this structure, enabling them to create interactive and responsive web applications. The unit covers various techniques for traversing the DOM tree, accessing and modifying elements, handling events, and updating content in real-time. By mastering DOM manipulation, developers can create dynamic web experiences that respond to user actions and inputs effectively. Furthermore, the unit explores best practices and common challenges associated with DOM manipulation. It emphasizes the importance of performance optimization and efficient code organization to ensure smooth user experiences, especially in complex applications. Additionally, the unit discusses cross-browser compatibility issues and techniques for writing code that works seamlessly across different web browsers. Understanding DOM manipulation is crucial for web developers as it forms the foundation for building interactive and engaging web applications, empowering

them to create dynamic content and enhance user interactions on the web.

5.0 References/Further Readings

Goodman, D. (2002). Dynamic HTML: The definitive reference: A comprehensive resource for HTML, CSS, DOM & JavaScript. " O'Reilly Media, Inc.".

Aggarwal, S. (2018). Modern web-development using reactjs. International Journal of Recent Research Aspects, 5(1), 133-137.

Harold, E. R., & Means, W. S. (2004). XML in a nutshell: a desktop quick reference. " O'Reilly Media, Inc.".

Maruyama, H. (2002). XML and Java: developing Web applications. Addison-Wesley Professional.

Dournaee, B., & Dournee, B. (2002). XML security (pp. 107-278). New York: Mcgraw-hill.