TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

TOPS Training Methodologies

June 2024

Learning Methodology

Trainer –

- Lecture Delivery
- Doubt Solving
- Assignment Support
- Internal Assessments
- Motivate to stay Focused

Student –

- Understanding of Topics – Presentations
- On time Assignment Practice
- Constant Revision and repetition
- Participate in all Assessments
- Regular in classes, practice and presentations

Sr. Trainer –

- Validate and discuss your performance
- Assign timelines for improvement
- Responsible for Course Success
- Go to Point for any concerns @center

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Resource Center

- CareerCenter
  - Demo for CareerCenter
- Recorded Video Links
- Assignment Document
- External Mocks
- Assessments (Internal)
- Placement Support
- Industry Tours
- Expert Talks

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Course and Profile Building Materials

Technical/Coding Students
- GitHub
- Projects
- Assignments Tasks

Designing Students
- Behance
- Portfolio

Digital Marketing Students
- Certifications

All Students
- Resume with Projects and Work Experience (if any)
- Linkedin Profile

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Career Courses Offered @TOPS Technologies

- Full Stack Courses (PHP, Python, .Net, .Net Core, Java, Node)
- Mern Stack (Node, React, Express, JS)
- FrontEnd (HTML, CSS, Bootstrap/Tailwind, JS,)
- Digital Marketing
- Software Testing
- Automation Testing
- Website Designing
- Graphic Desining
- UI/UX Designing
- Data Science, Data Analytics, Business Analytics, Machine Learning, Artificial Intelligence, Gen AI
- Networking, Security, Servers, Cyber Security, Ethical Hacking

TOPSTECHNOLOGIES

Training | Outsourcing | Placement | Study Abroad

Our TOPS Training Centers in India

Ahmedabad (C.G.Road)

+91 99747 55006 Get Direction

Ahmedabad (SG Road)

+91 99044 22211 Get Direction

Ahmedabad (Maninagar)

+91 99748 63333 Get Direction

Vadodara

+91 97253 55009 Get Direction

Rajkot

+91 97240 04242 Get Direction

Surat (Varachha)

+91 7383023957 Get Direction

Surat (Ring Road)

+91 70695 98828 Get Direction

Gandhinagar

+91 76240 06888 Get Direction

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

MernStack

Index

- SE(SE - Overview of IT Industry)
- Web Designing
- React Js
- Node Js

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module - 1 [Overview of IT Industry]

What is Program

- Program- It is a set of Instructions

Ex. 1- Instructions to your Pet Ex. 2- Starting your Computer

What is Programming

- Programming- To create a Program.

Ex. 1- Using Keyboard & Mouse

Types of Programming Language

- Procedural Programming
  - Ex.- C Language

Types of Programming Language

- Object Oriented Programming
  - Ex.- C++ Language

Types of Programming Language

- Logical Programming
  - Ex.- Prolog Language

Types of Programming Language

- Functional Programming
  - Ex.- Python Language

World Wide Web - WWW

- known as a Web
- is a collection of websites or web pages stored in web servers
- connected to local computers through the internet
- These websites contain text pages, digital images, audios, videos, etc.
- Users can access the content of these sites from any part of the world over the internet using their devices such as computers, laptops, cell phones, etc.
- The WWW, along with internet, enables the retrieval and display of text and media to your device.

WWW – How Internet Works?

- A server is where websites are stored, and it works a lot like your computer's hard drive.
- Once the request arrives, the server retrieves the website and sends the correct data back to your computer.
- One of the best features of the Internet is the ability to communicate almost instantly with anyone in the world.
- **Email** is one of the oldest and most universal ways to communicate and share information on the Internet, and billions of people use it. **Social media** allows people to connect in a variety of ways and build communities online.

WWW – How Internet Works?

Network layers on Client & Server

Client & Servers

1. **Client Program**

- Running on end host
- Requests Services
- E.g. Web Browser

Client & Servers

1. Client Program

- Running on end host
- Requests Services
- E.g. Web Browser

2. Server Program

- Running on end host
- Provides Services
- E.g. Webserver

Client – Servers Communication

1. Client "sometimes on"

- Initiates a request to the server when interested
- E.g. Web Browser on your laptop or cell phone
- Doesn't communicate directly with other clients
- Needs to know the server's address

2. Server is "always on"

- Services requests from many client hosts
- E.g.Web Server for the www.example.com web site
- Doesn't initiate contract with the clients
- Needs a fixed, well-known address

Types of Internet connection

1. Digital subscriber line(DSL)
2. Cable Internet
3. Fiber Optic
4. Satellite Internet
5. Wireless
6. Broadband over Power lines(BPL)

Major Applications of the Internet

- 4 major applications of internet, which are given below.

1. Social Media Internet Applications

   e.g .Facebook,Instagram,Twitter,LinkedIn…

2. Communications Internet Applications

   e.g. Email,Skype,Zoom,Whatsapp…

3. Entertainment Internet Applications

   e.g. Netflix,Hotstar,Youtube, Amazon prime video…

4. Travel Internet Applications

   e.g Google Trips,Google Map,trivago…

What are protocols?

- A Network Protocol is a group of rules accompanied by the network.
- Network protocols will be formalized requirements and plans composed of rules, procedures, and types that describe communication among a couple of devices over the network.
- The protocol can be described as an approach to rules that enable a couple of entities of a communication program to transfer information through any type of variety of a physical medium.
- The protocol identifies the rules, syntax, semantics, and synchronization of communication and feasible error managing methods. In this article, we will discuss the different types of networking protocols.

Types of Protocols

1. HTTP or HTTPS
2. FTP(File Transfer protocols)
3. Email Protocols(POP3,SMTP)
4. TCP(Transmission control protocol) and UDP(User Datagram Protocol)

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

What is Application Security?

- Application security refers to security precautions used at the application level to prevent the theft or hijacking of data or code within the application.
- It includes security concerns made during application development and design, as well as methods and procedures for protecting applications once they've been deployed.
- All tasks that introduce a secure software development life cycle to development teams are included in application security shortly known as AppSec.
- Its ultimate purpose is to improve security practices and, as a result, detect, repair, and, ideally, avoid security flaws in applications.
- It covers the entire application life cycle, including requirements analysis, design, implementation, testing, and maintenance..

What is Application Security?

- All tasks that introduce a secure software development life cycle to development teams are included in application security shortly known as AppSec.
- Its ultimate purpose is to improve security practices and, as a result, detect, repair, and, ideally, avoid security flaws in applications.
- It covers the entire application life cycle, including requirements analysis, design, implementation, testing, and maintenance..

Software Applications

What is Application Software :

- It is a type of software application that helps in the automation of the task based on the Users Input.
- It can perform single or multiple tasks at the same period of time.
- There are the different application which helps us in our daily life to process our instructions based on certain rules and regulations.
- Application Software helps in providing a graphical user interface to the user to operate the computer for different functionality.
- The user may use the computer for browsing the internet, accessing to email service, attending meetings, and playing games.
- Different high-level languages are used to build application software.

Types of Application Software

- Application software
- System software
- Driver software
- Middleware
- Programming software

Application Software

- The most common type of software, application software is a computer software package that performs a specific function for a user, or in some cases, for another application.
- An application can be self-contained, or it can be a group of programs that run the application for the user.
- Examples of Modern Applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms.

Example:Microsoft Office, Paint, Powerpoint etc..

System Software

- These software programs are designed to run a computer's application programs and hardware.
- - System software coordinates the activities and functions of the hardware and software.
- It controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in.
- The OS is the best example of system software; it manages all the other computer programs.
- Other examples of system software include the firmware, computer language translators and system utilities..

Example:Notepad ,Calculator etc..

Driver Software

- Also known as device drivers, this software is often considered a type of system software.
- Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks.
- Every device that is connected to a computer needs at least one device driver to function.
- Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers.

Example: Audio Driver,Video Driver etc..

Middleware

- The term *middleware* describes software that mediates between application and system software or between two different kinds of application software. For example, middleware enables Microsoft Windows to talk to Excel and Word.
- It is also used to send a remote work request from an application in a computer that has one kind of OS, to an application in a computer with a different OS. It also enables newer applications to work with legacy ones.

Example: database middleware,application server middleware

Programming Software

- Computer programmers use programming software to write code. Programming software and programming tools enable developers to develop, write, test and debug other software programs.
- Examples of programming software include assemblers, compilers, debuggers and interpreters.

Examples : Turbo c,Eclipse,Sublime etc..

Architecture

Applicaton and Examples

- Shopping mall Example
  1. Accept Customer Details
  2. calculate the bill
  3. Apply Discount Based on day of the week
     (a) Monday Low discount -10%
     (b) Thursday High discount-50%

- Shopping Mall

1. Accept customer detail
2. Calculate bill amount
3. Apply discount based on day of week
   1. Monday – Low discount - 10%
   2. Thursday – High discount – 50%

- Customer Detail
- CalculateBill
- GetFinalBill

Shopping Mall will contain discount logic.

Open closed principle - software entities should be open for extension, but closed for modification.

New discount strategy may be applied in future

Open closed principle – software entities should be open for extension, but closed for modification.

Shopping Mall
strategy
Discount Strategy
Discount algorithm
LowDiscountStrategy
Discount algorithm
HighDiscountStrategy
Discount algorithm
NoDiscountStrategy
Discount algorithm

Software Architecture

- Software architecture is the blueprint of building software. It shows the overall structure of the software, the collection of components in it, and how they interact with one another while hiding the implementation.

- This helps the software development team to clearly communicate how the software is going to be built as per the requirements of customers.
- There are various ways to organize the components in software architecture. And the different predefined organization of components in software architectures are known as software architecture patterns.

- A lot of patterns were tried and tested. Most of them have successfully solved various problems. In each pattern, the components are organized differently for solving a specific problem in software architectures.

- Well, I hope you don’t want to bore yourself by reading the endless types of software architecture patterns.

Layers in Software Architecture

1. Presentation layer
2. Application layer
3. Business layer
4. Persistence layer
5. Database layer

Presentation
Application
Business
Persistence
Database
Calls between layers flow downwards

1. Presentation layer

The presentation layer, also called the UI layer, handles the interactions that users have with the software. It's the most visible layer and defines the application's overall look and presentation to the end-users. This is the tier that's most accessible, which anyone can use from their client device, like a desktop, laptop, mobile phone or tablet.

2. Application layer

The application layer handles the main programs of the architecture. It includes the code definitions and most basic functions of the developed application. This is the layer that programmers spend most of their time in when working on the software. You can use this layer to implement specific coordination logic that doesn't align exactly with either the presentation or business layer.

3. Business layer

The business layer, also called the domain layer, is where the application's business logic operates. Business logic is a collection of rules that tell the system how to run an application, based on the organization's guidelines. This layer essentially determines the behavior of the entire application. After one action finishes, it tells the application what to do next.

4. Persistence layer

The persistence layer, also called the data access layer, acts as a protective layer. It contains the code that's necessary to access the database layer. This layer also holds the set of codes that allow you to manipulate various aspects of the database, such as connection details and SQL statements.

5. Database layer

The database layer is where the system stores all the data. It's the lowest tier in the software architecture and houses not only data but indexes and tables as well. Search, insert, update and delete operations occur here frequently. Application data can store in a file server or database server, revealing crucial data but keeping data storage and retrieval procedures hidden.

Environments in industry

There are different Types of environments in Industry.

1. The analysis and design environment
2. The development environment
3. The common build environment
4. The testing environment
5. The production environment

Environments in industry

1. Analysis & Design Environment

- The analysis and design environment is aligned to the **planning and analysis phases** of the SDLC. In this environment, the main processes that take place include carrying out an in-depth examination of the current system and the proposed system. The system architecture is also defined and includes developing the design of the hardware, software, and network requirements for the system. Within this environment, systems and business analysts work closely with software engineers.

2. The development environment

- The development environment can also be a physical space where development takes place and where software engineers interact. Another example of the development environment is the **integrated development environment** (IDE). The IDE provides a platform where tools and development processes are coordinated in order to provide software engineers a convenient way of accessing the resources they require during the development process.

Environments in industry

3. The common build environment

- The **common build environment** is closely aligned to the development phase of the SDLC. In this environment, software engineers merge the work done in the development environment. Within this environment, software engineers build systems. These are used to automate the process of software compilation.

4. The testing environment

- The test environment is where testing teams evaluate the application/quality. program's This also allows computer programmers to find out and solve any defects that may interfere with the application's smooth operation or degrade the user experience.

5. The production environment

- When the end-user use a web/mobile application, the program is operating on a production server. It's been created in the production environment.

SDLC

- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software.

SDLC Methodology

- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:

1. Requirement Gathering
2. Analysis
3. Designing
4. Implementation
5. Testing
6. Maintenance

Programming

There are countless definitions of what computer programming is, but here is Ours.

"Programming is how *you* get computers to solve problems."

There are two key phrases here that are important:

- **You**: without the programmer (you), the computer is useless. It does what **you** tell it to do.
- **Solve problems**: computers are tools. They are complex tools, admittedly, but they are not mysterious or magical: they exist to make tasks easier.

Programming

Computer programs make computers work

Computer programs (or software) are what makes computers work. Without software, modern computers are just complicated machines for turning electricity into heat. It’s software on your computer that runs your operating system, browser, email, games, movie player – just about everything.

Programming

Programming is creative:

Programming is a creative task: there is no right or wrong way to solve a problem, in the same way, that there is no right or wrong way to paint a picture.

There are choices to be made, and one way may seem better than another, but that doesn’t mean the other is wrong! With the right skills and experience, a programmer can craft software to solve an unlimited number of problems – from telling you when your next train will arrive at playing your favourite music.

The possibilities are constrained only by your imagination. That’s why I love programming.Programming having a multiple Programming Languages.

Programming Languages

- A **programming language** is a computer language programmers use to develop software programs, scripts, or other sets of instructions for computers to execute.
- Although many languages share similarities, each has its own syntax. Once a programmer learns the languages rules, syntax, and structure, they write the source code in a text editor or IDE. Then, the programmer often compiles the code into machine language that can be understood by the computer. Scripting languages, which do not require a compiler, use an interpreter to execute the script.
- A programming language is a set of instructions that can be used to interact with and control a computer. These languages are used to design websites, create apps, develop operating systems, control spacecraft, and analyze data. Programming languages are necessary because computers can't understand English. Programming languages bridge this gap by helping programmers translate their commands into something that the computer can understand and execute.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Types of Programing Languages

Types of Programing Languages

1. Low-level programming language

Low-level language is **machine-dependent (0s and 1s)** programming language. The processor runs low- level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.

Low-level language is further divided into two parts -

i. Machine Language

Machine language is a type of low-level programming language. It is also called as **machine code or object code**. Machine language is easier to read because it is normally displayed in binary or hexadecimal form (base 16) form. It does not require a translator to convert the programs because computers directly understand the machine language programs.

The advantage of machine language is that it helps the programmer to execute the programs faster than the high-level programming language.

ii. Assembly Language

Assembly language (ASM) is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a **symbolic and human-understandable form**. It uses an assembler to convert the assembly language to machine language.

The advantage of assembly language is that it requires less memory and less execution time to execute a program.

2. High-level programming language

High-level programming language (HLL) is designed for **developing user-friendly software programs and websites**. This programming language requires a compiler or interpreter to translate the program into machine language (execute the program).

The main advantage of a high-level language is that it is **easy to read, write, and maintain**.

High-level programming language includes **Python, Java, JavaScript, PHP, C#, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN, and Swift programming language**.

A high-level language is further divided into three parts -

i. Procedural Oriented programming language

Procedural Oriented Programming (POP) language is derived from structured programming and based upon the procedure call concept. It divides a program into small procedures called **routines or functions**.

Procedural Oriented programming language is used by a software programmer to create a program that can be accomplished by using a programming editor like IDE, Adobe Dreamweaver, or Microsoft Visual Studio.

The advantage of POP language is that it helps programmers to easily track the program flow and code can be reused in different parts of the program.

Example: C, FORTRAN, Basic, Pascal, etc.

ii. Object-Oriented Programming language

Object-Oriented Programming (OOP) language is **based upon the objects**. In this **programming language, programs are divided into small parts called objects**. It is used to implement real-world entities like inheritance, polymorphism, abstraction, etc in the program to makes the program resusable, efficient, and easy-to-use.

The main advantage of object-oriented programming is that OOP is faster and easier to execute, maintain, modify, as well as debug.

Example: C++, Java, Python, C#, etc.

iii. Natural language

Natural language is a **part of human languages** such as English, Russian, German, and Japanese. It is used by machines to understand, manipulate, and interpret human's language. It is used by developers to **perform tasks such as translation, automatic summarization, Named Entity Recognition (NER), relationship extraction, and topic segmentation**.

The main advantage of natural language is that it helps users to ask questions in any subject and directly respond within seconds.

3. Middle-level programming language

Middle-level programming language **lies between the low-level programming language and high-level programming language**. It is also known as the intermediate programming language and pseudo-language.

A middle-level programming language's advantages are that it supports the features of high-level programming, it is a user-friendly language, and closely related to machine language and human language.

Example: C, C++, language

Important Programming Language

- **Development is Mainly Divided into two Parts**

Front end Development

HTML

CSS

JAVASCRIPT

React

Angular

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Important Programming Language

Backend Development

In Backend you have a many languages you can choose anyone language

If you are Choosing web development platform then you have a Mainly Three Languages

PHP

JAVA

PYTHON

Dot Net

Important Programming Language

HTML:

HTML—“HyperText Markup Language”—is **the language used to tell your web browser what each part of a website is**. So, using HTML, you can define headers, paragraphs, links, images, and more, so your browser knows how to structure the web page you're looking at.

CSS:

CSS makes the front-end of a website shine and it creates a great user experience. Without CSS, websites would be less pleasing to the eye and likely much harder to navigate. In addition to layout and format, CSS is responsible for font color and more.

Important Programming Language

JAVASCRIPT:

JavaScript has evolved over the past 25 years **to become a versatile and accessible programming language for working with web browsers**. Developers use JavaScript to build complex interactive websites and browser games, and to connect servers to websites and web applications

Important Programming Language

The backend (or “server-side”) is the portion of the website you don't see. It's **responsible for storing and organizing data, and ensuring everything on the client-side actually works**. The backend communicates with the frontend, sending and receiving information to be displayed as a web page.

Writing Source Code

What is Source Code?

Source code is the source of a computer program. It contains declarations, instructions, functions, loops and other statements, which act as instructions for the program on how to function. Programs may contain one or more source code text files, which can be stored on a computer's hard disk, in a database, or be printed in books of code snippets.

Important Notes For Source Code

- Coding Must Have Com m ents
- Coding Must Have a Prop er Architecture
- Coding Must Have a
- Declaring Coding Versions
- Divide your code in packages

Running Code

When you will run your code two things are Most Important

1.Compiler

2.Interpreter

1.Compiler:

It is a translator which takes input i.e., High-Level Language, and produces an output of low-level language i.e. machine or assembly language.

- A compiler is more intelligent than an assembler it checks all kinds of limits, ranges, errors, etc.

- But its program run time is more and occupies a larger part of memory. It has a slow speed because a compiler goes through the entire program and then translates the entire program into machine codes.

2. Interpreter :

An interpreter is a program that translates a programming language into a comprehensible language. –

- It translates only one statement of the program at a time.
- Interpreters, more often than not are smaller than compilers.

GitHub

- GitHub is one of the most popular resources for developers to share code and work on projects together. It’s free, easy to use, and has become central in the movement toward open-source software.
- Git is used for managing the changes to a project over time. A project might be just a single file, a handful of files, or thousands of files. Those files can be anything from plain text to images or videos.
- Because Git is focused on managing changes, it is often used as a collaboration tool allowing people to work on the same project at the same time. By tracking their individual changes, Git can bring everything together to the final version.

Working with Git

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Website Designing

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Modules

Module 1 - [HTML]

Module 2 - [CSS and CSS3]

Module 3 - [HTML5]

Module 4 - [Bootstrap Basic And Advanced]

Module 5 - [Javascript]

Module 6 - [React JS]

Module 7 - [Node JS]

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 1)
Mernstack - HTML

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

- ✓ Introduction to HTML
- ✓ HTML Elements
- ✓ HTML Attribute
- ✓ HTML Basic Tags
- ✓ HTML Doctypes
- ✓ HTML Layout
- ✓ HTML Head
- ✓ HTML Meta
- ✓ HTML Scripts
- ✓ HTML Comments
- ✓ HTML Images
- ✓ HTML Tables
- ✓ HTML Lists
- ✓ Text Links
- ✓ Image Links
- ✓ HTML Headings
- ✓ HTML Paragraphs
- ✓ Text Formatting
- ✓ HTML Iframes
- ✓ HTML Multimedia

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Hypertext Transfer Protocol

The Hypertext Transfer Protocol is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems

https://github.com/Brijesh1990/tops-website- developmen t/tree/master/introductionWD/protocol/http

Text Editor

Notepad

Sublime Text

What is HTML

HTML stands for Hypertext Markup Language.

HTML is used to create web pages and web applications.

HTML is widely used language on the web.

We can create a static website by HTML only.

Technically, HTML is a Markup language rather than a programming language.

https://github.com/Brijesh1990/tops-website-development/tree/master/introductionWD/introduction-html

Introduction of html

- HTML is the standard markup language for creating Web pages.
- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading",
- "This is a paragraph", "this is a link", etc.

- The `<!DOCTYPE html>` declaration defines that this document is an HTML5 document
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

HTML Tags

HTML tags are like keywords which defines that how web browser will format and display the content.

With the help of tags, a web browser can distinguish between an HTML content and a simple content.

HTML tags contain three main parts: opening tag, content and closing tag.

But some HTML tags are unclosed tags.

Rules for Tags

All HTML tags must enclosed within < > these brackets. Every tag in HTML perform different tasks.

If you have used an open tag <tag>, then you must use a close tag </tag> (except some tags)

Ex.   ,

Example 1

Write an HTML program to print “Hello World”.

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>Hello world.</p>
</body>
</html>
```

HTML Meta Tags

<!DOCTYPE html>
<title>
<link>
<meta>
<style>

HTML Text Tags

<p>
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>

<cite>

HTML Unclosed Tags

<hr>

HTML Element

These elements are responsible for creating web pages and define content in that webpage.

An element is a collection of start tag, attributes, end tag, content between them.

Void Element

Void element: All the elements in HTML do not require to have start tag and end tag, some elements does not have content and end tag such elements are known as Void elements or empty elements. These elements are also called as unpaired tag.

Ex.

<hr>

Block-level element

These are the elements, which structure main part of webpage, by dividing a page into coherent blocks.

A block-level element always start with new line and takes the full width of web page, from left to right.

These elements can contain block-level as well as inline elements

Ex.
<h1>-<h6>
<hr>

Inline elements:

- Inline elements are those elements, which differentiate the part of give text and provide it a particular function.

- These elements does not start with new line and take width as per requirement.

- The Inline elements are mostly used with other elements.

Ex.

```
<a>
<label>
<b>
```

Useful HTML Elements

HTML Link
Tags HTML
Image HTML
List HTML
Table HTML
Form

Attributes

HTML attributes are special words which provide additional information about the elements or attributes are the modifier of the HTML element.

Each element or tag can have attributes, which defines the behaviour of that element.

Attributes should always be applied with start tag.

The Attribute should always be applied with its name and value pair.

The Attributes name and values are case sensitive, and it is recommended by W3C that it should be written in Lowercase only.

You can add multiple attributes in one HTML element, but need to give space between two attributes.

Syntax

<element attribute_name="value">content</element>

Ex.
<body text="green" bgcolor="orange">

Ex.
<h1 title="This is heading tag">Example of title attribute</h1>

Text Formatting

What is Formatting

HTML Formatting is a process of formatting text for better look and feel. HTML provides us ability to format text without using CSS.

Categories:

- Physical tag: These tags are used to provide the visual appearance to the text.
- Logical tag: These tags are used to add some logical or semantic value to the text.

Formatting Text

1. Bold Text: <b> and
2. Italic Text: <i > and
3. Marked Formatting: <mark>
4. Underlined Text: <u> and <ins>
5. Strike Text: <strike> and <del>
6. Monospaced Font: <tt>
7. Superscript Text: <sup>
8. Subscript Text: <sub>
9. Larger Text: <big>
10. Smaller Text: <small>

HTML Phrase tag

The HTML phrase tags are special purpose tags, which defines the structural meaning of a block of text or semantics of text.

Phrase Tags Ex.

1. Abbreviation: <abbr title = ""> </abbr>
2. Marked: <mark>
3. Strong:
4. Emphasized:
5. Definition: <dfn>
6. Quoting: <blockquote cite="">, <cite>
7. Short: <q>
8. Code: <code>
9. Keyboard: <kbd>
10. Address: <address>

HTML Comments

Syntax:

```
<!--
Write commented text
here
-->
```

```
<html lang="en"> event
▶ <head> ··· </head>
▼ <body>
  ▶ <style> ··· </style>
    <!--
    The text in here will be invisible on the website! Here's
    another line of the comment. You can have as many lines as you
    want! 😁
    -->
  ▼ <div class="content">
      And here's that regular HTML content again.
    </div>
  </body>
</html>
```

HTML HEAD

Head

The HTML <head> element is used as a container for metadata (data about data). It is used between <html> tag and <body> tag.

The head of an HTML document is a part whose content is not displayed in the browser on page loading. It just contains metadata about the HTML document which specifies data about the HTML document.

Tags

Following is a list of tags used in metadata:

<title>
<link>
<style>
<script>
<meta>
<base>

<meta>

```
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta http-equiv="refresh" content="30">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

<base>

<base href="https://www.learnvern.com/images/" target="_blank">

https://github.com/Brijesh1990/tops-website-development/tree/master/module1- html/head-tags

HTML Layout

Layout

Tags

<header>: It is used to define a header for a document or a section.
<nav>: It is used to define a container for navigation links
<section>: It is used to define a section in a document
<article>: It is used to define an independent self-contained article
<aside>: It is used to define content aside from the content (like a sidebar)
<footer>: It is used to define a footer for a document or a section
<details>: It is used to define additional details
<summary>: It is used to define a heading for the <details> element

HTML Entities

Entities

HTML character entities are used as a replacement of reserved characters in HTML. You can also replace characters that are not present on your keyboard by entities.

For example: if you use less than (<) or greater than (>) symbols in your text, the browser can mix them with tags that's why character entities are used in HTML to display reserved characters.

Syntax

&entity_name;
OR
&#entity_number;

Example

<	non-breaking space	
	less than	<
>	greater than	>
&	ampersand	&
"	double quotation mark	"
'	single quotation mark (apostrophe)	'
¢	cent	¢
£	pound	£
¥	yen	¥
€	Euro	€
©	copyright	©
®	registered trademark	®

Symbols

There are many mathematical, technical and currency symbols which are not present on a normal keyboard. We have to use HTML entity names to add such symbols to an HTML page.

If there no entity name exists, you can use an entity number, a decimal, or hexadecimal reference.

Ex.

Α	Α	Α	GREEK CAPITAL LETTER ALPHA
Β	Β	Β	GREEK CAPITAL LETTER BETA
Γ	Γ	Γ	GREEK CAPITAL LETTER GAMMA
Δ	Δ	Δ	GREEK CAPITAL LETTER DELTA
Ε	Ε	Ε	GREEK CAPITAL LETTER EPSILON
Ζ	Ζ	Ζ	GREEK CAPITAL LETTER ZETA
←	←	←	LEFTWARDS ARROW
↑	↑	↑	UPWARDS ARROW
→	→	→	RIGHTWARDS ARROW
↓	↓	↓	DOWNWARDS ARROW

Find GitHub links

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body- tags/html-entity

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body- tags/html-entity

HTML Table

HTML Table

HTML table tag is used to display data in tabular form (row * column). There can be many columns in a row.

HTML tables are used to manage the layout of the page e.g. header section, navigation bar, body content, footer section etc. But it is recommended to use div tag over table to manage the layout of the page .

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/table

Example:

```
<table>
        <tr>
    <th>1 header</th>
    <th>1 header</th>
    <th>1 header</th>
  </tr>
        <tr>
    <td>1data</td>
    <td>1data</td>
    <td>1data</td>
  </tr>
      </table>
```

Table Supportive tags

<table>

<tr>

<th>

<td>

<caption>

<tbody>

<thead>

<tfooter>

Attributes:

1. rowspan
2. colspan

HTML List

HTML Lists

HTML Lists are used to specify lists of information. All lists may contain one or more list elements. There are three different types of HTML lists:

- Ordered List or Numbered List (ol)
- Unordered List or Bulleted List (ul)
- Description List or Definition List (dl)

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/html-list

Ordered List or Numbered List

All the list items are marked with numbers by default.

```
<ol>
 <li>Aries</li>
 <li>Bingo</li>
 <li>Leo</li>
 <li>Oracle</li>
</ol>
```

Types:
1, I, i, A, a

Unordered List or Bulleted List

All the list items are marked with bullets.

```
<ul>
 <li>Aries</li>
 <li>Bingo</li>
 <li>Leo</li>
 <li>Oracle</li>
</ul>
```

Types:
disc, circle, square, none

Description List or Definition List

Entries are listed like a dictionary or encyclopedia.

```
<dl>
  <dt>HTML</dt>
  <dd>is a markup language</dd>
  <dt>Java</dt>
  <dd>is a programming language and
  platform</dd>
 <dt>JavaScript</dt>
 <dd>is a scripting language</dd>
  <dt>SQL</dt>
  <dd>is a query language</dd>
</dl>
```

HTML File Path and Iframe Tag

File Paths

An HTML file path is used to describe the location of a file in a website folder. Used to link images, file, CSS file, JS file, video, etc.

Attributes:

1. src

Ex.

1. <img src="picture.jpg">
2. <img src="images/picture.jpg">
3. <img src="/images/picture.jpg">
4. <img src="../picture.jpg">

Types of File Paths

1. Absolute File Paths:
Absolute file path specifies full URL address. Ex.

```
<img src="https://www.learnvern.com/"
alt="image" style="width:300px">
```

2. Relative File Paths

```
<img src="images/learnvern.jpg" alt="Mountain" style="width:300px">
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/html-iframe

IFrames

HTML Iframe is used to display a nested webpage (a webpage within a webpage). Used to embed Webpage or a YouTube video.

Syntax:<iframe src="URL"></iframe>

Attributes:

1. src
2. width
3. height
4. frameborder
5. allowfullscreen

HTML Form

HTML Form

An HTML form is a section of a document which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc.

An HTML form facilitates the user to enter data that is to be sent to the server for processing such as name, email address, password, phone number, etc.

HTML forms are required if you want to collect some data from of the site visitor.

Ex. Online Shopping Website

Syntax

```
<form action="server url"
method="get/post">

</form>
```

HTML Form Elements

<form>
<input>
<textarea>
<label>
<fieldset>
<legend>

Input Types

```
<input type="text" name="username">
<input type="password" name="password">
<input type="email" name="email">
<input type="radio" name="password">
<input type="checkbox" id="cricket" name="cricket" value="cricket"/>
<input type="submit" value="submit">
<input type="button" value="button">
```

Form Inputs Types

Input Types

text:	Defines a one-line text input field
password:	Defines a one-line password input field
submit :	Defines a submit button to submit the form to server
reset:	Defines a reset button to reset all values in the form.
radio:	Defines a radio button which allows select one option.
checkbox:	Defines checkboxes which allow select multiple options form.
button:	Defines a simple push button, which can be programmed to perform a task on an event.
file:	Defines to select the file from device storage.
image:	Defines a graphical submit button.

HTML5 Added Input Types

color:	Defines an input field with a specific color.
date:	Defines an input field for selection of date.
datetime-local:	Defines an input field for entering a date without time zone.
email:	Defines an input field for entering an email address. Defines a control with month and year, without time zone. Defines an input field to enter a number.
month:	
number:	
url:	Defines a field for entering URL
week:	Defines a field to enter the date with week-year, without time zone.
search:	Defines a single line text field for entering a search string.
tel:	Defines an input field for entering the telephone number.

HTML Form Attributes

Form Attributes

1.action: The action attribute value defines the web page where information proceed. It can be .php, .jsp, .asp, etc. or any URL

1. method: Defines the HTTP method

- post: We can use the post value of method attribute when we want to process the sensitive data as it does not display the submitted data in URL.
- get: The get value of method attribute is default value while submitting the form. But this is not secure as it displays data in URL after submitting the form.

1. target: Where to open the response after submitting the form

- _self: The response will display in current page only.
- _blank: Load the response in a new page.

Form Attributes

3. autocomplete: Enables an input field to complete automatically.

3. enctype: Defines the encoding type

- application/x-www-form-urlencoded: Default
- multipart/form-data: It does not encode any character. It is used when our form contains file-upload controls.
- text/plain (HTML5): only space are encoded into + symbol

5. novalidate: Does not perform any type of validation and submit the form.

Input Attributes

1. name
2. value
3. required
4. autofocus
5. placeholder
6. disabled
7. size
8. form

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/form

Class and Id

Class

The HTML class attribute is used to specify a single or multiple class names for an HTML element.

The class name can be used by CSS and JavaScript to do some tasks for HTML elements.

You can use this class in CSS with a specific class, write a period (.) character, followed by the name of the class for selecting elements.

ID

The id attribute is used to specify the unique ID for an element of the HTML document.

It allocates the unique identifier which is used by the CSS and the JavaScript for performing certain tasks.

https://github.com/Brijesh1990/tops-website-development/tree/master/module2-css%26css3-basic-advanced/css/Selector

HTML Form and Keyboard Event Attributes

What is an Event?

When a browser reacts on user action, then it is called as an event.

Form Event Attributes

Attribute	Description
onblur	Executed the script when form element loses the focus.
onchange	Executed the script when the value of the element is changed.
onfocus	Trigger an event when the element gets focused.
oninput	Executed the script when the user enters input to the element.
oninvalid	Executed the script when the element does not satisfy its predefined constraints.
onreset	Triggers the event when user reset the form element values.
onsubmit	Triggers the event when a form is submitted.

HTML Keyboard and Mouse Event Attributes

Keyboard Event Attributes

Attribute	Description
onkeydown	Triggers the event when the user presses down a key on the keyboard.
onkeypress	Trigger the event when the user presses the key which displays some character.
onkeyup	Trigger the event when the user releases the currently pressed key.

Mouse Event Attributes

Attribute	Description
onclick	Trigger the event when the mouse clicks on the element.
ondblclick	Trigger the event when mouse double-click occurs on the element.
onmousedown	Trigger the event when the mouse button is pressed on the element.
onmousemove	Trigger the event when the mouse pointer moves over the element.
onmouseout	Trigger the event when the mouse moves outside the element.
onmouseover	Trigger the event when the mouse moves onto the element.
onmouseup	Trigger the event when the mouse button is released.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 2) Mernstack – CSS and CSS 3

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

CSS

- ✓ Introduction to CSS
- ✓ How to insert CSS
- ✓ Inline
- ✓ Internal
- ✓ External
- ✓ Comments in CSS
- ✓ CSS Selectors
- ✓ CSS Pseudo Selector
- ✓ CSS Text
- ✓ CSS Fonts
- ✓ CSS Background
- ✓ CSS Display and Position Properties
- ✓ CSS Buttons

- ✓ CSS Colors
- ✓ CSS Important
- ✓ Line height, Padding and Margin
- ✓ CSS Width and Height
- ✓ CSS Box-shadow and Text-shadow
- ✓ CSS Visibility
- ✓ CSS Selectors
- ✓ CSS Specificity
- ✓ CSS Positions

Advance CSS

- ✓ CSS Pseudo elements
- ✓ CSS Gradient
- ✓ CSS Combinators
- ✓ CSS Media Query
- ✓ CSS Flex
- ✓ CSS Grid

What is CSS

CSS stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.

CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

What is CSS

- CSS stands for Cascading Style Sheet.
- CSS is used to design HTML tags.
- CSS is a widely used language on the web.
- HTML, CSS and JavaScript are used for web designing. It helps the web designers to apply style on HTML tags.

Advantages of CSS

- Solves a big problem
- Saves a lot of time
- Provide more attributes

CSS Syntax

declaration

↓

selector → p{color: black}

↑ ↑

property value

Selector{Property1: value1; Property2: value2; ..... ;}

CSS Syntax

Selector: Selector indicates the HTML element you want to style. It could be any tag like <h1>, <title> etc.

Declaration Block: The declaration block can contain one or more declarations separated by a semicolon.

Property: A Property is a type of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

How to add CSS

```
<body>
<h1>Write Your First CSS Example</h1>
<p>This is Paragraph.</p>
</body>
```

How to add CSS

```
<head>
<style>
h1{ color:white;
background-color:red;
padding:5px;
}
p{ color:blue;
}
</style>
</head>
```

CSS Selector

CSS Selector

CSS selectors are used to select the content you want to style. Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

Types of CSS Selector:

- CSS Universal Selector
- CSS Element Selector
- CSS Id Selector
- CSS Class Selector
- CSS Group Selector

CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

Ex.

```
<style>
* {
  color: green; font-
  size: 20px;
}
</style>
<h2>This is heading</h2>
<p>This style will be applied on every paragraph.</p>
<p id="para1">Me too!</p>
<p>And me!</p>
```

CSS Element Selector

The element selector selects the HTML element by name.

Ex.

```
<style> p{
   text-align: center;
   color: blue;
}
</style>

<p>This style will be applied on every paragraph.</p>
```

CSS Id Selector

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.

It is written with the hash character (#), followed by the id of the element.

Ex.

```
<style>
#para1 {
   text-align: center;
   color: blue;
}
</style>
<p id="para1">Hello Javatpoint.com</p>
```

CSS Class Selector for specific element

If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector.

Ex.

```
<style>
p.center {
    text-align: center;
    color: blue;
}
</style>
<h1 class="center">This heading is not affected</h1>
<p class="center">This paragraph is blue and center-aligned.</p>
```

CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.
Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

Ex.

```
<style> h1, h2, p
{
   text-align: center; color:
   blue;
}
</style>
```

CSS Combinators

CSS Combinators

CSS Combinators clarifies the relationship between two selectors, whereas the selectors in CSS are used to select the content for styling.

There can be more than one simple selector in a CSS selector, and between these selectors, we can include a combinator. Combinators combine the selectors to provide them a useful relationship and the position of content in the document.

There are four types of combinators in CSS that are listed as follows:

1. General sibling selector (~)
2. Adjacent sibling selector (+)
3. Child selector (>)
4. Descendant selector (space)

General Sibling Selector

It uses the tlide (~) sign as the separator between the elements. It selects the elements that follow the elements of first selector, and both of them are the children of the same parent. It can be used for selecting the group of elements that share the common parent element.

It is useful when we have to select the siblings of an element even if they are not adjacent directly.

Syntax:

```
element ~ element {
   /*style properties*/
}
```

Adjacent Sibling Selector (+)

It uses the plus (+) sign as the separator between the elements. It matches the second element only when the element immediately follows the first element, and both of them are the children of the same parent. This sibling selector selects the adjacent element, or we can say that the element which is next to the specified tag.

It only selects the element which is just next to the specified first element.

Syntax:

```
element + element {
  /*style properties*/
}
```

Child Selector (>)

It uses the greater than (>) sign as the separator between the elements. It selects the direct descendant of the parent. This combinator only matches the elements that are the immediate child in the document tree. It is stricter as compared to the descendant selector because it selects the second selector only when the first selector is its parent.

The parent element must always be placed at the left of the ">". If we remove the greater than (>) symbol that designates this as a child combinator, then it will become the descendant selector.

Syntax:

```
element > element {
   /*style properties*/
}
```

Descendant Selector (space)

It combines two selectors in which the first selector represents an ancestor (parent, parent's parent, etc.), and the second selector represents descendants. The elements matched by the second selector are selected if they have an ancestor element that matches the first selector.

Syntax:

```
element element {
  /*style properties*/
}
```

CSS Psudo Selector

CSS pseudo-classes

A pseudo-class can be defined as a keyword which is combined to a selector that defines the special state of the selected elements. It is added to the selector for adding an effect to the existing elements based on their states. For example, The ":hover" is used for adding special effects to an element when the user moves the cursor over the element.

Syntax:

```
selector: pseudo-class
  { property: value;
}
```

:hover

This pseudo-class adds a special style to an element when the user moves the cursor over it.

Ex.

```
h1:hover
    {
    color:red;
}
```

:active

It applies when the elements are clicked or activated. It selects the activated element.

Ex.

```
a:active{
        color: yellow;
    }
```

:visited

It selects the visited links and adds special styles to them.

Ex.

```
a:visited{
        color: red;
    }
```

:lang

It is helpful in documents that require multiple languages.

Ex.

```
p:lang(fr)
   {
      font-family:Verdana; color:blue;

   }

<p lang="fr">With :lang pseudo class with the value fr</p>
```

:focus

It selects the elements that are currently focused on by the user.

Ex.

```
input:focus{
        border:5px solid lightblue;
        box-shadow:10px 10px 10px
        black; color: blue;
        width:300px;
        }
```

:first-child

It matches a particular element, which is the first child of another element and adds a special effect to the corresponding element.

Ex.

```
h1:first-child {
        text-indent:
         200px; color:blue;
}
```

:nth-child(n)

This selector is used for matching the elements based on their position regardless of the type of its parent. The n can either be a keyword, formula, or a number. It is used to match the elements based on their position within a group of siblings. It matches each element, which is the nth-child.

Ex.

```
p:nth-child(2n+1) {
        background:
        yellow; color: black;
        font-size:30px;
      }
```

CSS Pseudo Element

CSS Pseudo-elements

A pseudo-class can be defined as a keyword which is combined to a selector that defines the special state of the selected elements. Unlike the pseudo-classes, the pseudo-elements are used to style the specific part of an element, whereas the pseudo-classes are used to style the element.

Syntax:

```
selector::pseudo-element
  { property: value;
}
```

We have used the double colon notation (::pseudo-element) in the syntax. In CSS3, the double colon replaced the single colon notation for pseudo-elements.

Psuedo Elements

Psuedo Element	Description
::first-letter (:first-letter)	It selects the first letter of the text.
::first-line (:first-line)	It styles the first line of the text.
::before (:before)	It is used to add something before the element's content.
::after (:after)	It is used to add something after the element's content.
::selection	It is used to select the area of an element that is selected by the user.

::first-letter

it affects the first letter of the text. It can be applied only to block-level elements. Instead of supporting all CSS properties, it supports some of the CSS properties that are given below.

- Color properties (such as color)
- Font properties (such as font-style, font-family, font-size, font-color, and many more).
- Margin properties (such as margin-top, margin-right, margin-bottom, and margin-left).
- Border properties (like border-top, border-right, border-bottom, border-left, border-color, border-width, and many more).
- Padding properties (such as padding-top, padding-right, padding-bottom, and padding-left).
- Background properties (such as background-color, background-repeat, background-image, and background-position).
- Text related properties (such as text-shadow, text-transform, text-decoration, etc.).
- Other properties are vertical-align (only when the float is 'none') word-spacing, line-height, line-spacing, etc.

::first-line

It is similar to the ::first-letter pseudo-element, but it affects the entire line. It adds the special effects to the first line of the text. It supports the following CSS properties:

- Color properties (such as color)
- Font properties (such as font-style, font-family, font-size, font-color, and many more).
- Background properties (such as background-color, background-repeat, background-image, and background-position).
- Other properties are word-spacing, letter-spacing, line-height, vertical-align, text-transform, text-decoration.

::before

It allows us to add something before the element's content. It is used to add something before the specific part of an element. Generally, it is used with the content property.

Ex:

```
h1::before {
   content: "'Hello World.'";
}
```

::after

It works similar to ::before pseudo-element, but it inserts the content after the content of the element. It is used to add something after the specific part of an element. Generally, it is used with the content property.

Ex:

```
h1::after {
   content: "'Welcome to the LearnVern'";
}
```

::selection

It is used to style the part of an element that is selected by the user. We can use the following CSS properties with it:

- color.
- background-color.
- Other properties include cursor, outline, etc.

Ex.

```
h1::selection
   { color: red;
}
```

How to add CSS

How to add CSS

There are three ways to insert CSS in HTML documents.

1. Inline CSS
2. Internal CSS
3. External CSS

Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

Syntax

```
<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>
```

Ex.

```
<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>
<p>This paragraph is not affected.</p>
```

https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/1cssInline.html

Disadvantages of Inline CSS

- You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.
- These styles cannot be reused anywhere else.
- These styles are tough to be edited because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in <head> section of the HTML page inside the <style> tag.

Ex.

```
<style>
body {
    background-color: linen;
}
h1 {
    color: red; margin-left: 80px;
}
</style>
```

External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Ex.

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

The external style sheet may be written in any text editor but must be savedwith a .css extension. This file should not contain HTML elements.

CSS Comments

Need of the comments

CSS comments are generally written to explain your code. It is very helpful for the users who reads your code so that they can easily understand the code.

Comments are ignored by browsers.

Comments

Comments are single or multiple lines statement and written within /*.....*/ .

Ex.

```
<style>
p {
   color: blue;
   /* This is a single-line comment
   */ text-align: center;
}
```

```
 /* This is
a multi-line
comment */
</style>
```

Specificity

Specificity

Specificity is the way which helps the browsers to decide which property value is most relevant for the element. It determines which style declaration is applied to an element.

- The CSS specificity is important only when various selectors are affecting the same element. In this case, the browser needs a way to identify the style to be applied to the matching element, and CSS specificity is the way of doing it.
- When two or more selectors have equal specificity value, then the latest one considers.
- Universal selectors (*) and the inherited values have lower specificity, i.e., 0 specificity.
- The style property has a greater specificity value compare to the selectors (except the !important in the stylesheet selector).
- The !important alter the selector specificity. When two selectors have equal specificity, then the selector having !important

hierarchy

Inline styles: It is directly attached to the element which is to be styled. For example: <p style="color: red;">. It has the highest priority.

IDs: It is a unique identifier for the elements of a page that has the second-highest priority. For example: #para.

Classes, attributes, and pseudo-classes: It includes classes, attributes, and pseudo- classes (like :focus, :hover, etc.).

Elements and pseudo-elements: It includes the name of elements (div, h1) and pseudo-elements (like :after and :before). They have the lowest priority.

Rules

- The specificity of ID selectors is higher than attribute selectors
- In equal specificity, the latest rule will count
- The specificity of class selector is greater than the element selectors

CSS Fonts

CSS Font property is used to control the look of texts. By the use of CSS font property you can change the text size, color, style and more.

- CSS Font color
- CSS Font family
- CSS Font size
- CSS Font style
- CSS Font variant
- CSS Font weight

CSS Font Color

It is used to change the color of the text. There are three different formats to define a color:

By a color name By hexadecimal value By RGB

Ex.

```
h1 { color: red; }
h2 { color: #9000A1; }
p { color:rgb(0, 220, 98); }
```

CSS Font Family

Generic family: It includes Serif, Sans-serif, and Monospace.
Font family: It specifies the font family name like Arial, New Times Roman etc.

Serif: Serif fonts include small lines at the end of characters. Example of serif: Times new roman, Georgia etc.

Sans-serif: A sans-serif font doesn't include the small lines at the end of characters. Example of Sans-serif: Arial, Verdana etc.

CSS Font Family

Ex.

```
h1 { font-family: sans-serif;
} h2 { font-family: serif; }
p { font-family: monospace;
}
```

CSS Font Size

CSS font size property is used to change the size of the font.

font-size:xx-small;
font-size:x-small;
font-size:small;
font-size:medium;
font-size:large;
font-size:x-large;
font-size:xx-large;
font-size:smaller;
font-size:larger;
font-size:200%;
font-size:20px;

CSS Font Size

1. Absolute-size:

It is used to set the text to a definite size. Using absolute-size, it is not possible to change the size of the text in all browsers. It is advantageous when we know the physical size of the output.

Ex. Font-size with em

2. Relative-size:

It is used to set the size of the text relative to its neighboring elements.
With relative-size, it is possible to change the size of the text in browsers.

CSS Font Size

3. Responsive font size:

We can set the size of the text by using a vw unit, which stands for the 'viewport width'. The viewport is the size of the browser window.

Ex. 1vw = 1% of viewport width.

4. Font-size with the length property:

It is used to set the size of the font in length. The length can be in cm, px, pt, etc.

Ex. font-size: 5cm;

CSS Font Style

CSS Font style property defines what type of font you want to display. It may be italic, oblique, or normal.

Ex.

```
h2 { font-style: italic; }
h3 { font-style: oblique;
}
h4 { font-style: normal; }
```

CSS Font Variant

CSS font variant property specifies how to set font variant of an element. It may be normal and small-caps.

Ex.

```
p { font-variant: small-caps;
} h3 { font-variant: normal; }
```

CSS Font Weight

CSS font weight property defines the weight of the font and specify that how bold a font is. The possible values of font weight may be normal, bold, bolder, lighter or number (100, 200..... upto 900).

Ex.

font-weight:bold;
font-weight:bolder;
font-weight:lighter;
font-weight:100;
font-weight:200;
font-weight:300;
font-weight:900;

CSS Font Stretch

The font-stretch property in CSS allows us to select a normal, expanded, or condensed face from the font's family. This property sets the text wider or narrower compare to the default width of the font. It will not work on any font but only works on the font-family that has a width-variant face.

Ex.

font-stretch: normal / semi-condensed / condensed / extra-condensed / ultra- condensed / semi-expanded/ expanded / extra-expanded / ultra-expanded

CSS Background

CSS Background

CSS background property is used to define the background effects on element. There are 5 CSS background properties that affects the HTML elements:

1. background-color
2. background-image
3. background-repeat
4. background-attachment
5. background-position

1. CSS background-color

The background-color property is used to specify the background color of the element.

Ex.

```
<style>
p{
   background-color: green;
}
</style>
```

2. CSS background-image

The background-image property is used to set an image as a background of an element. By default the image covers the entire element

Ex.

```
<style>
body {
background-image:
url("image.jpg"); margin-left:100px;
}
</style>
```

3. CSS background-repeat

By default, the background-image property repeats the background image horizontally and vertically. Some images are repeated only horizontally or vertically.

Ex.

```
<style> body {
    background-image: url("gradient_bg.png");
    background-repeat: repeat-x;        or        background-repeat: repeat-y;
}
</style>
```

4. CSS background-attachment

The background-attachment property is used to specify if the background image is fixed or scroll with the rest of the page in browser window. If you set fixed the background image then the image will not move during scrolling in the browser.

Ex.

```
<style>
background: white
url('image.jpg'); background-
repeat: no-repeat;
background-attachment: fixed;
</style>
```

5. CSS background-position

The background-position property is used to define the initial position of the background image. By default, the background image is placed on the top-left of the webpage.

You can set the positions as:center,top,bottom,left,right

Ex.

```
background: white url('good-morning.jpg');
background-repeat: no-repeat;
background-attachment: fixed;
background-position: center;
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module2-css%26css3-basic-advanced/css/background

CSS Border Property

CSS Border

The CSS border is a shorthand property used to set the border on an element.

The CSS border properties are use to specify the style, color and size of the border of an element. The CSS border properties are given below

- border-style
- border-color
- border-width
- border-radius

1. CSS Border Style

The Border style property is used to specify the border type which you want to display on the web page.

Ex.

border-style: none;
border-style: dotted;
border-style: dashed; border-style: solid; border-style: double;

border-style: groove; border-style: ridge; border- style: inset; border- style: outset; border-style: hidden

2. CSS Border Width

The border-width property is used to set the border's width. It is set in pixels. You can also use the one of the three pre-defined values, thin, medium or thick to set the width of the border.

Ex.

```
<style>
p.one {
   border-style: solid;
   border-width: 5px;  / border-width: medium; / border-width: 1px;
}
</style>
```

3. CSS Border Color

There are three methods to set the color of the border. Name: It specifies the color name. For example: "red".
RGB: It specifies the RGB value of the color. For example: "rgb(255,0,0)".
Hex: It specifies the hex value of the color. For example: "#ff0000“.

Ex.

```
p.one {
   border-style: solid;
   border-color: red;
}
p.two {
```

4. CSS Border Radius

This CSS property sets the rounded borders and provides the rounded corners around an element, tags, or div. It defines the radius of the corners of an element. The values of this property can be defined in percentage or length units.
It is shorthand for border top-left-radius, border-top-right-radius, border-bottom- right-radius and border-bottom-left-radius.

Ex.

Variations are as below:

Single Value - border-radius: 30px;

Two Value - border-radius: 20% 10%;

5. CSS Border Collapse

This CSS property is used to set the border of the table cells and specifies whether the table cells share the separate or common border.

This property has two main values that are separate and collapse. When it is set to the value separate, the distance between the cells can be defined using the border-spacing property. When the border-collapse is set to the value collapse, then the inset value of border-style property behaves like groove, and the outset value behaves like ridge.

Ex.

```
table {
border-collapse: separate;      / border-collapse: collapse;
}
```

6. CSS Border Spacing

This CSS property is used to set the distance between the borders of the adjacent cells in the table. It applies only when the border-collapse property is set to separate. There will not be any space between the borders if the border-collapse is set to collapse.

When only one value is specified, then it sets both horizontal and vertical spacing. When we use the two-value syntax, then the first one is used to set the horizontal spacing (i.e., the space between the adjacent columns), and the second value sets the vertical spacing (i.e., the space between the adjacent rows).

Ex.

table {

border-collapse:

separate; border-spacing:

7. CSS Outline

CSS outline is just like CSS border property. It facilitates you to draw an extra border around an element to get visual attention.

Ex.

```
.Outline {
    background-color: #eee;
    border: 3px solid
    Lightgreen; padding: 5px
    10px;
    outline-width: 3px;
    outline-style: solid;
    outline-color: red;
```

Outline offset: The outline offset is used to create a distance between outline and border.

7. CSS Outline...

Outline offset: The outline offset is used to create a distance between outline and border.

.Outline-
Offset
{ background-
color:
#eee; outline: 3px solid
red; outline-offset: 6px;
border: 3px solid
Lightgreen; padding: 5px

8. CSS Border Image

This CSS property defines an image to be used as the element's border. It draws an image outside the element and replaces the element's border with the corresponding image. It is an interesting task to replace the border of an element with the image.

It is the shorthand property for border-image-source, border-image-slice, border- image-width, border-image-outset, and border-image-repeat.

Ex.

```
border-image: url('border.png') 60 / 20px 20px round;
```

https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/border.html

8. CSS Border Image...

Value	Description
border-image-source	It specifies the source of the border-image.
border-image-slice	It is used to divide or slice the image, which is specified by the border- image-source property.
border-image-width	It sets the width of the border-image.
border-image-outset	It sets the amount of space by which the border image is set out from its border box.
border-image-repeat	It controls the repetition of the image to fill the area of the border. stretch repeat round space

CSS Display Property

CSS Display

CSS display is the most important property of CSS which is used to control the layout of the element. It specifies how the element is displayed.

CSS Display Value:

- display: inline;
- display: inline-block;
- display: block;
- display: none;
- flex

1. Inline

The inline element takes the required width only. It doesn't force the line break so the flow of text doesn't break in inline example.

The inline elements are:

<a>

<b> etc.

Ex.

```
p {
display: inline;
}
```

2. inline-block

The CSS display inline-block element is very similar to inline element but the difference is that you are able to set the width and height.

Ex.

```
p {
display: inline-block;
}
```

3. block

The CSS display block element takes as much as horizontal space as they can. Means the block element takes the full available width. They make a line break before and after them.

Ex.

```
p {
display: block;
}
```

4. none

The "none" value totally removes the element from the page. It will not take any space.

Ex.

hidden

{ display:

none;

}

5. flex

It is used to display an element as an block-level flex container. It is new in css3.

Ex.

```
p {
display: flex;
}
```

https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/advance/14Advance5Display.html

https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/advance/12Advance5.1Display.html

CSS Flex

CSS Flexbox Layout Module

Before the Flexbox Layout module, there were four layout modes:

- ✓ Block, for sections in a webpage
- ✓ Inline, for text
- ✓ Table, for two-dimensional table data
- ✓ Positioned, for explicit position of an element
- ✓ The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

CSS Flex Items

The direct child elements of a flex container automatically becomes flexible (flex) items.

```
<style>
.flex-container
  { display: flex;
  background-color: #f1f1f1;
}
.flex-container > div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>
```

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container
  { display: flex;
  background-color: DodgerBlue;}
.flex-container > div
  { background-color:
  #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;}
</style>
</head>
<body>
<h1>Create a Flex Container</h1>
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
</body>
</html>
```

CSS Grid

Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  </div>
```

CSS Grid Container

To make an HTML element behave as a grid container, you have to set the display property to grid or inline-grid.

```
<style>
.grid-container
  { display: grid;
  grid-template-columns: auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
```

CSS Grid Item

A grid *container* contains grid *items*.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

Note: The grid-column property is a shorthand property for the grid-column-start and the grid-column-end properties.

Example

Make "item1" start on column 1 and end before column 5:

```
.item1 {
  grid-column: 1 / 5;
}
```

CSS Buttons

CSS Button

In HTML, we use the button tag to create a button, but by using CSS properties, we can style the buttons. Buttons help us to create user interaction and event processing.

background-color

This property is used for setting the background color of the button element.

Ex.

```
button {
        color:lightgoldenrodyellow;
}
```

border

It is used to set the border of the button. It is the shorthand property for border- width, border-color, and border-style.

Ex.

```
button {
        border:none;   / border:5px brown solid; / border: 5px red dashed;
/ border: 5px   black dotted;   / border:5px blue double;
}
```

border-radius

It is used to make the rounded corners of the button. It sets the border radius of the button.

Ex.

```
button {
        border-radius: value in px or %;
}
```

box-shadow

As its name implies, it is used to create the shadow of the button box. It is used to add the shadow to the button. We can also create a shadow during the hover on the button.

box-shadow: [horizontal offset] [vertical offset] [blur radius] [optional spread radius] [color];

Ex.

```
button {
}           box-shadow : 0 8px 16px 0 black,     0 6px 20px 0 rgba(0, 0, 0, 0.19);
button :hover{
        box-shadow : 0 8px 16px 0              0 6px 20px 0 rgba(0, 0, 0, 0.19);
        black,
    }
```

padding

It is used to set the button padding.

box-shadow: [horizontal offset] [vertical offset] [blur radius] [optional spread radius] [color];

Ex.

```
button {
        padding: value in px;
}
```

CSS Positioning

CSS Float

The CSS float property is a positioning property. It is used to push an element to the left or right, allowing other element to wrap around it. It is generally used with images and layouts.

Center Aligned

Left Aligned

Right Aligned

Concept of Float

A floated element may be moved as far to the left or the right as possible. Simply, it means that a floated element can display at extreme left or extreme right.

The elements after the floating element will flow around it.

The elements before the floating element will not be

affected.

If the image floated to the right, the texts flow around it, to the left and if the image floated to the left, the text flows around it, to the right.

Float Example

```
<style>
img {
   float: right;
}
</style>
```

https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/advance/15Advance6.1Relativeposition.html

CSS clearfix

A clear float (or clearfix) is a way for an element to fix or clear the child elements so that we do not require to add additional markup. It resolves the error which occurs when more than one floated elements are stacked next to each other.

Ex.

```
p
{
clear:right;
}
```

CSS Colors

CSS Colors

The color property in CSS is used to set the color of HTML elements. Typically, this property is used to set the background color or the font color of an element.

Ways to show colors in CSS:

1. RGB format.
2. RGBA format.
3. Hexadecimal notation.
4. HSL.
5. HSLA.
6. Built-in color.

RGB Format

RGB format is the short form of 'RED GREEN and BLUE' that is used for defining the color of an HTML element simply by specifying the values of R, G, B that are in the range of 0 to 255.

The color values in this format are specified by using the rgb() property. This property allows three values that can either be in percentage or integer (range from 0 to 255).

This property is not supported in all browsers;

Syntax:

color: rgb(R, G, B);

RGBA Format

It is almost similar to RGB format except that RGBA contains A (Alpha) that specifies the element's transparency. The value of alpha is in the range 0.0 to 1.0, in which 0.0 is for fully transparent, and 1.0 is for not transparent.

Syntax:

color:rgba(R, G, B, A);

Hexadecimal notation

Hexadecimal can be defined as a six-digit color representation. This notation starts with the # symbol followed by six characters ranges from 0 to F. In hexadecimal notation, the first two digits represent the red (RR) color value, the next two digits represent the green (GG) color value, and the last two digits represent the blue (BB) color value.

Syntax:

color:#(0-F)(0-F)(0-F)(0-F)(0-F)(0-F);

Shorthand Hex-Code:

It is a short form of hexadecimal notation in which every digit is recreated to arrive at an equivalent hexadecimal value.

For example, #7B6 becomes #77BB66 in hexadecimal.

HSL

It is a short form of Hue, Saturation, and Lightness. Let's understand them individually.

Hue: It can be defined as the degree on the color wheel from 0 to 360. 0 represents red, 120 represents green, 240 represents blue.

Saturation: It takes value in percentage in which 100% represents fully saturated, i.e., no shades of gray, 50% represent 50% gray, but the color is still visible, and 0% represents fully unsaturated, i.e., completely gray, and the color is invisible.

Lightness: The lightness of the color can be defined as the light that we want to provide the color in which 0% represents black (there is no light), 50% represents neither dark nor light, and 100% represents white (full lightness).

HSLA

It is entirely similar to HSL property, except that it contains A (alpha) that specifies the element's transparency. The value of alpha is in the range 0.0 to 1.0, in which 0.0 indicates fully transparent, and 1.0 indicates not transparent.

Syntax:

color:hsla(H, S, L, A);

Built-in Color

As its name implies, built-in color means the collection of previously defined colors that are used by using a name such as red, blue, green, etc.

Syntax:
color: color-name;

CSS Margin

Ex.

margin: value auto;

margin: value length in px,pt

cm; margin: value %;

Margin Shorthand properties:

margin: 50px 100px 150px 200px; margin: 50px 100px 150px; margin: 50px 100px; margin 50px;

CSS Padding

CSS Padding property is used to define the space between the element content and the element border.

It is different from CSS margin in the way that CSS margin defines the space around elements. CSS padding is affected by the background colors. It clears an area around the content.

Top, bottom, left and right padding can be changed independently using separate properties. You can also change all properties at once by using shorthand padding property.

CSS Padding Properties

padding
padding-left
padding-right
pdding-top
padding-bottom

Values:
padding: value in %;
padding: value in length as in px, pt, cm;

Margin Shorthand properties:
padding: 50px 100px 150px 200px; padding: 50px 100px 150px; padding: 50px 100px; padding: 50px;

CSS Images

Images are an important part of any web application. Including a lot of images in a web application is generally not recommended, but it is important to use the images wherever they required. CSS helps us to control the display of images in web applications.

The styling of an image in CSS is similar to the styling of an element by using the borders and margins. There are multiple CSS properties such as border property, height property, width property, etc. that helps us to style an image.

Thumbnail Image

The border property is used to make a thumbnail image.

Ex.

```
img{
border: 2px solid
red; border-
radius:5px;
padding:10px;
}
```

Transparent image

To make an image transparent, we have to use the opacity property. The value of this property lies between 0.0 to 1.0, respectively.

Ex.

```
img{
border: 2px solid red;
border-radius:5px;
padding:10px;
opacity:0.3;
}
```

Rounded image

The border-radius property sets the radius of the bordered image. It is used to create the rounded images.

Ex.

```
#img1{
border-radius:10px;
}

#img2{
border-radius:50%;
}
```

Responsive Image

It automatically adjusts to fit on the screen size. It is used to adjust the image to the specified box automatically.

Ex.

```
#img1{
max-width:100%;
height:auto;
}
```

Center an Image

We can center an image by using the left-margin and right-margin property. We have to set these properties to auto in order to make a block element.

Ex.

```
#img1{
margin-left:auto;
margin-right:auto;
display:block;
}
```

CSS Width and Height

CSS Width Property

The CSS width property is used to set the width of the content area of an element.

It does not include padding borders or margins. It sets width of the area inside the padding, border, and margin of the element.

Values:

width: auto;

width:

length;

width: %;

We can use max-width and min-width properties to set the maximum and minimum width of

CSS Height Property

This CSS property sets the height of an element. It is used to set the height of content area of an element.

It does not include padding borders or margins, whereas it sets the height of the area inside the padding, border, and margin of the element. It can accept the length and percentage values. But it does not allow negative values.

If we set the height to a numeric value (like in px, %, etc.), the content can be overflow if it does not fit in the given height. We can manage the overflowing content by defining the overflow property.

CSS Height Values

Values:

width: auto;

width: length;

We can use **max-height and min-height** properties to set the maximum and minimum height of the element

CSS Box-shadow and Text-shadow

Box-shadow CSS

It is used to add shadow-like effects around the frame of an element.

Syntax:

box-shadow: h-offset v-offset blur spread color |inset|inherit|initial|none;

Text-shadow CSS

As its name implies, this CSS property adds shadows to the text. It accepts the comma-separated list of shadows that applied to the text. It's default property is none. It applies one or more than one text-shadow effect on the element's text content.

Syntax:

text-shadow: h-shadow v-shadow blur-radius color| none | initial | inherit;

Box-shadow/Text Shadow Values

h-offset: It horizontally sets the shadow position. Its positive value will set the shadow to the right side of the box. Its negative value is used to set the shadow on the left side of the box.

v-offset: Unlike the h-offset, it is used to set the shadow position vertically. The positive value in it sets the shadow below the box, and the negative value sets the shadow above of the box.

blur: As its name implies, it is used to blur the box-shadow. This attribute is optional.

spread: It sets the shadow size. The spread size depends upon the spread value.

color: As its name implies, this attribute is used to set the color of the shadow. It is an optional attribute.

Box-shadow/Text Shadow Values

inset: Normally, the shadow generates outside of the box, but by using inset, the shadow can be created within the box.

initial: It is used to set the property of the box-shadow to its default

value. inherit: it is inherited from its parent.

none: It is the default value that does not include any shadow property.

Box-shadow Ex.

Ex.

h-offset, v-offset and blur attributes box-shadow: 5px 10px 10px;

spread attribute
box-shadow: 5px 10px 10px 10px;

color attribute
box-shadow: 5px 10px 10px 10px orange;

inset attribute
box-shadow: 5px 10px 10px 10px orange inset;

initial attribute
box-shadow: initial;

default attribute
box-shadow: none;

Box-shadow Ex.

Ex.
initial attribute
box-shadow: initial;

default attribute
box-shadow:
none;

Text-shadow

Ex.

Simple Shadow

text-shadow: 3px 3px red;

Fuzzy Shadow

text-shadow: 3px 3px 3px violet;

Multiple Shadows

text-shadow: -3px -3px 3px blue, 3px 3px 3px red;

Glow Effect

text-shadow: 0 0 .1em cyan;

CSS Visibility

CSS Visibility

The CSS visibility property is used to specify whether an element is visible or not.

Note: An invisible element also take up the space on the page. By using display property you can create invisible elements that don't take up space.

Syntax:
visibility: visible|hidden

Ex. with CSS

```
h1.visible
   { visibility:
   visible
}
h1.hidden
   { visibility:
   hidden
}
```

Ex. with javascrpit

```
function myFunction()
{ if(document.getElementById("myDIV").style.visibility ==
"visible"){ document.getElementById("myDIV").style.visibility =
"hidden";
}
else{
document.getElementById("myDIV").style.visibility = "visible";
}
```

CSS Icons

CSS Icons

Icons can be defined as the images or symbols used in any computer interface refer to an element. It is a graphical representation of a file or program that helps the user to identify about the type of file quickly.

Using the icon library is the easiest way to add icons to our HTML page. It is possible to format the library icons by using CSS. We can customize the icons according to their color, shadow, size, etc.

There are given some of the icon libraries such as Bootstrap icons, Font Awesome icons, and Google icons that can be used in CSS easily. There is no need to install or download the libraries mentioned above.

Font Awesome Icons

To use this library in our HTML page, we need to add the following link within the
<head></head> section.

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
```

```
<i class="fa fa-cloud"></i>
<i class="fa fa-file"></i>
<i class="fa fa-heart"></i>
<i class="fa fa-bars"></i>
<i class="fa fa-car"</i>
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/font-aswome

Bootstrap Icons

As similar to the font awesome library, we can add the bootstrap icons in our HTML page using the link given below in the <head></head> section.

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
>
    <i class="glyphicon glyphicon-cloud"></i>
    <i class="glyphicon glyphicon-file"></i>
    <i class="glyphicon glyphicon-heart"></i>
    <i class="glyphicon glyphicon-user"></i>
    <i class="glyphicon glyphicon-thumbs-up"></i>
    <i class="glyphicon glyphicon-remove"></i>
    <i class="glyphicon glyphicon-envelope"></i>
```

Google Icons

As similar to the above libraries, we can add it in our HTML page simply by adding the link given below in the <head></head> section.

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons"
>

    <i class="material-icons">cloud</i>
    <i class="material-icons">attachment</i>
    <i class="material-icons">computer</i>
    <i class="material-icons">favorite</i>
    <i class="material-icons">traffic</i>
```

Justify, Text Decoration and Text-align

Justify

This CSS property is used to align the items of the flexible box container when the items do not use all available space on the main-axis (horizontally). It defines how the browser distributes the space around and between the content items.

This CSS property can't be used to describe containers or items along the vertical axis. To align the items vertically, we have to use the align-items property.

Syntax:

justify-content: center | flex-start | flex-end | space-around | space-evenly | space-between

Property Values

center: As its name implies, it set the position of items at the center of the container.
flex-start: It is the default value that positioned the items at the beginning of the container.

flex-end: It set the position of items at the end of the container.

space-around: It positioned the items with equal spacing from each other. It evenly distributes the items in the line along with the same space around them.

space-between: Items are evenly spaced in which the first item is at the beginning, and the last item is at the end.

space-evenly: It also positioned the items with equal space, but the spacing from the corners differs.

Text Decoration

It is a CSS property that decorates the content of the text. It adds lines under, above, and through the text. It sets the appearance of decorative lines on text. This CSS property decorates the text with several kinds of lines. This is shorthand for text-decoration-line, text-decoration-color, and text-decoration-style.

Text Decoration can be done in below mentioned ways:

1. text-decoration-line
2. text-decoration-color
3. text-decoration-style

text-decoration-line

It sets the kind of text-decoration like overline, underline, or line-through. It can be used to add a combination of lines to the selected text.

Syntax:

text-decoration: underline | line-through |

overline text-decoration: overline underline line-

through;

text-decoration-style

This property is used to set the style of the line. Its values are solid, dotted, wavy, double, and dashed.

Syntax:

text-decoration: underline double| line-through dashed| overline

overline text-decoration: lightblue wavy overline underline line-through;

text-decoration-color

This property is used to provide color to the decoration. Its value is any color in a valid format.

Syntax:

text-decoration: underline double cyan| line-through dashed green|

dotted overline blue

text-decoration: lightblue wavy overline underline line-through;

Text Align

This CSS property is used to set the horizontal alignment of a table-cell box or the block element. It is similar to the vertical-align property but in the horizontal direction.

Syntax:
text-align: justify | center | right | left

Property Values

justify: It is generally used in newspapers and magazines. It stretches the element's content in order to display the equal width of every line.

center: It centers the inline text.

right: It is used to align the text to the right.

left: It is used to align the text to the left.

:nth-child(n) selector and :first-child selector

nth-child(n) selector

This selector is used for matching the elements based on their position regardless of the type of its parent. The n can either be a keyword, formula, or a number. It is used to match the elements based on their position within a group of siblings. It matches each element, which is the nth-child.

Syntax:

```
:nth-child(n) {

}
```

The "n" in the parentheses is the argument that represents the pattern for matching elements. It can be a functional notation, even or odd.

Ex.

1. Using Function:

```
<style>
        p:nth-child(2n+1) {
         }
</style>
```

2.Using even and odd.

```
p:nth-child(even) {
         }

p:nth-child(odd) {       }
```

first-child and last-child Selector

The :first-child selector is used to select the specified selector, only if it is the first child of its parent.

Syntax:

```
:first-child {

}
```

The :last-child selector matches every element that is the last child of its parent.

Syntax:

```
p:last-child {

}
```

CSS Print Properties

CSS Print

While we need to print the current webpage, css prints page related properties comes into the picture.

There are total of 3 types of print properties available.

1. page-break-before
2. page-break-inside
3. page-break-after

page-break-before

As the name implies, this CSS property is used to add the page break before the element, when printing the document. It inserts a page break before the specified element during printing the document. We cannot use this CSS property on absolutely positioned elements or an empty element that does not generate a box.

Syntax:

page-break-before: auto | always | left | right | avoid

Values

Value	Description
auto	It is the default value that inserts a page break before the element, if necessary.
always	This value always forces a page break before the specified element.
avoid	It is used to avoid a page break before the element whenever possible.
left	This value forces either one or two page breaks before the element so that the next page will be depicted as the left-hand page.
right	The right value forces either one or two page breaks before the element so that the next page will be depicted as the right-hand page.

page-break-inside

this CSS property is used to specify the page break inside the element, when printing the document. This CSS property cannot be used on absolutely positioned elements or an empty element that does not generate a box. It inserts or avoids the page break inside the specified element during printing the document.

Syntax:

page-break-inside: auto | avoid

Values

Value	Description
auto	It is the default value that inserts a page break inside the element, if necessary.
always	It is used to avoid a page break inside the element whenever possible.

page-break-after

This CSS property is used to adjust the page break after the element when printing the document. It inserts a page break after the specified element during printing.

Syntax:
page-break-after: auto | always | left | right | avoid

Values

Value	Description
auto	It is the default value that inserts a page break after the element, if necessary.
always	It always forces a page break after the specified element.
avoid	It is used to avoid a page break after the element whenever possible.
left	It forces either one or two page breaks after the specified element so that the next page will be depicted as the left-hand page.
right	It forces either one or two page breaks after the specified element so that the next page will be depicted as the right-hand page.

CSS Columns

CSS Columns

The columns property in CSS sets the number and width of columns in a single declaration. It is a shorthand property that can take several values at a time.

It is used to set both column-count and column-width properties at the same time. Both of these properties are used to control how many columns will appear. The column-count property is used to set the number of columns, whereas the column- width property specifies the width of the columns.

Together using column-count and column-width properties creates a multi-column layout that breaks automatically into a single column at narrow browser widths without using the media queries. It is not always helpful to use both of them because it can restrict the responsiveness and flexibility of the layout.

Values

Syntax:

columns: auto | column-width column-count|

Value	Description
Auto	It is the default value which sets the values of column-count and column-width to the default browser values.
column-width	It is used to set the minimum width for columns. However, the actual width of the column may be narrower or wider based on the available space.
column-count	It specifies the maximum number of columns.

Ex.

```
element name/selector{
        columns: (length) (column number);
}
```

Transition Delay

CSS Transition Delay Property

This CSS property specifies the duration to start the transition effect. The value of this property is defined as either **seconds (s) or milliseconds (ms)**. The CSS transitions are effects that are added to change the element gradually from one style to another, without using flash or JavaScript.
Without using the **transition-delay**, the animation will start with the hover on the element, but using this CSS property, we can delay the animation by an amount of time.

The default value of the transition-delay property is 0, which means that the transition will start to occur immediately without any delay.
Syntax:
transition-delay: time

Values

Value	Description
time	It specifies the amount of time (in seconds or milliseconds) to wait before the transition starts.
initial	It sets this property to its default value.
inherit	It inherits this property from its parent element.

Note

The negative value of the transition-delay property will immediately start the transition effect i.e., the effect will be animated as though it had already begun. The positive value of this property causes the transition effect to start for the given time.

Ex. 1

- Change the color of the div on hover

```
div{
width: 100px;
height:
100px;
background: lightblue;
transition-property: background-
color; transition-duration: 1s;
transition-delay: 0.5s;
/* For Safari browser */
-webkit-transition-property:
background- color;
-webkit-transition-duration: 1s;
-webkit-transition-delay: 0.5s;
}
div:hover {
background-color: brown;
```

Ex. 2

- Change the width of div on hover

```
.first{
width: 150px;
height: 150px;
background-color: lightblue;
transition-property: width;
transition-duration: 1s;
transition-delay: initial;

.first:hover
{ width:
300px;
}
```

Ex. 3

- Rotate the div on hover

```
padding:15px;
width:  200px;
height: 200px;
background:  lightgreen;
transition:  background-color 1s, width 2s, height 2s, transform 2s; transition-delay: 1.5ms;
```

```
div:hover
{ width:
300px;
height:
300px;
-webkit-transform:
rotate(360deg);
 /* Chrome, Safari, Opera
*/ transform:
rotate(360deg);
```

CSS Gradient

CSS Gradient and its uses

CSS gradient is used to display smooth transition within two or more specified colors.

- You don't have to use images to display transition effects.
- The download time and bandwidth usage can also be reduced.
- It provides better look to the element when zoomed, because the gradient is generated by the browser.

There are two types of gradients in CSS3

1. Linear gradients
2. Radial gradients

CSS Linear Gradient

The CSS3 linear gradient goes up/down/left/right and diagonally. To create a CSS3 linear gradient, you must have to define two or more color stops. The color stops are the colors which are used to create a smooth transition. Starting point and direction can also be added along with the gradient effect.

Syntax:

background: linear-gradient (direction, color-stop1, color-stop2. ...);

CSS Linear Gradient: (Top to Bottom)

Top to Bottom Linear Gradient is the default linear gradient. Let's take an example of linear gradient that starts from top. It starts red and transitions to green.

Ex.

```
#grad1 {
    height: 100px;
    background: -webkit-linear-gradient(red, green); /* For Safari 5.1 to 6.0 */
    background: -o-linear-gradient(red, green); /* For Opera 11.1 to 12.0 */
    background: -moz-linear-gradient(red, green); /* For Firefox 3.6 to 15 */
    background: linear-gradient(red, green); /* Standard syntax (must be last)
    */
```

CSS Linear Gradient: (Left to Right)

The following example shows the linear gradient that starts from left and goes to right. It starts red from left side and transitioning to green.

Ex.

```
#grad1 {
   height: 100px;
   background: -webkit-linear-gradient(left, red, green); /* For Safari 5.1 to 6.0
   */
   background: -o-linear-gradient(right, red, green); /* For Opera 11.1 to 12.0 */
   background: -moz-linear-gradient(right, red, green); /* For Firefox 3.6 to 15 */
   background: linear-gradient(to right, red , green); /* Standard syntax (must be
last) */
}
```

CSS Linear Gradient: (Diagonal)

If you specify both the horizontal and vertical starting positions, you can make a linear gradient diagonal.

Ex.

```
#grad1 {
   height: 100px;
   background: -webkit-linear-gradient(left top, red , green); /* For Safari 5.1 to 6.0 */
   background: -o-linear-gradient(bottom right, red, green); /* For Opera 11.1 to 12.0 */
   background: -moz-linear-gradient(bottom right, red, green); /* For Firefox 3.6 to 15 */
   background: linear-gradient(to bottom right, red , green); /* Standard syntax (must be last)
   */
}
```

CSS Radial Gradient

You must have to define at least two color stops to create a radial gradient. It is defined by its center.

Syntax:

background: radial-gradient(shape size at position, start-color, ..., last-color);

CSS Radial Gradient: (Evenly Spaced Color Stops)

Evenly spaced color stops is a by default radial gradient. Its by default shape is eclipse, size is farthest- carner, and position is center.

Ex.

```
#grad1 {
    height: 150px;
    width: 200px;
    background: -webkit-radial-gradient(blue, green, red); /* Safari 5.1 to 6.0 */
    background: -o-radial-gradient(blue, green, red); /* For Opera 11.6 to 12.0 */
    background: -moz-radial-gradient(blue, green, red); /* For Firefox 3.6 to 15 */
    background: radial-gradient(blue, green, red); /* Standard syntax (must be last)
    */
```

Radial Gradient: (Differently Spaced Color Stops)

Ex.

```
#grad1 {
   height: 150px;
   width: 200px;
   background: -webkit-radial-gradient(blue 5%, green 15%, red 60%); /* Safari 5.1 to 6.0 */
   background: -o-radial-gradient(blue 5%, green 15%, red 60%); /* For Opera 11.6 to 12.0 */
   background: -moz-radial-gradient(blue 5%, green 15%, red 60%); /* For Firefox 3.6 to 15 */
   background: radial-gradient(blue 5%, green 15%, red 60%); /* Standard syntax (must be last)
   */
}
```

Radial Gradient: (From Corners)

Ex.

```
#grad1 {
        background-image: radial-gradient(farthest-side at left bottom, red, yellow, cyan );
}
#grad2{
        background-image: radial-gradient(farthest-corner at right bottom ,blue, yellow,green);
}
#grad3{
        background-image: radial-gradient(closest-side , red, yellow, lightgreen);
}
#grad4{
        background-image: radial-gradient(closest-corner , blue, yellow, green);
}
```

CSS z-index

z-index

The z-index in CSS allows us to define a visual hierarchy on the 3-dimensional plane, i.e., the z-axis. It is used to specify the stacking order of the positioned elements (elements whose position value is either fixed, absolute, relative, or sticky). The stacking order means that the element's position along the z-axis, which is perpendicular to the screen.

Syntax:
z-index: number|auto

Ex.

```
div{
  position:fixed; z-index:20;
}
img{
  position:relative; z-index:22;
}
h1{
  position:relative; z-index:28;
```

Masking

CSS Masking

The mask property in CSS is used to hide an element using the clipping or masking the image at specific points. Masking defines how to use an image or the graphical element as a luminance or alpha mask. It is a graphical operation that can fully or partially hide the portions of an element or object.

Using masking, it is possible to show or hide the parts of an image with different opacity levels. In CSS, the masking is achieved by using the mask-image property, and we have to provide an image as the mask.

Ex.

Ex. 1: Mask

```
#mask{
-webkit-mask-box-image: url(box.png);
}
```

Ex. 2: Gradient

```
#mask{
-webkit-mask-image: -webkit-gradient(linear, right top, right
bottom, from(rgba(0,0,0,0)), to(rgba(0,0,0,0.9)));      |
-webkit-mask-image: radial-gradient(circle at 50% 50%, blue 40%,
rgba(0,0,0,0.3) 70%);
border: 9px ridge red;
}
```

CSS Media Query

What is Media Query?

CSS Media query is a W3C recommendation and a CSS3 module which is used to adapt to conditions such as screen resolution (e.g. Smartphone screen vs. computer screen).

- The media query technique first used in CSS3.
- It became a W3C recommendation in June 2012.
- It is an extension of media dependent stylesheets used in different media types (i.e. screen and print) found in CSS2.
- The most commonly used media feature is "width".
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

What is Responsive Web Design?

It facilitates you to use fluid grids, flexible images, and media queries to progressively enhance a web page for different viewing contexts i.e. Desktop, Smartphone, Tablet etc.

different screen resolutions

Smartphone: 320px

Tablet: 768px

Netbook: 1024px

Desktop: 1600px

Large Desktop: 1920px

Ex.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<style> body {
   background-color:yellow;
}

@media only screen and (max-width: 500px)
   { body {
      background-color:green;
   }
}
</style>
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 3) Mernstack - HTML5

- ✓ HTML5 Tags
- ✓ HTML Audio and Video Tag
- ✓ HTML SVG
- ✓ Canvas
- ✓ Display Grid

HTML Audio Tag

HTML Audio Tag

HTML audio tag is used to define sounds such as music and other audio clips. Currently there are three supported file format for HTML 5 audio tag.

- mp3
- wav
- ogg

HTML5 supports <video> and <audio> controls. The Flash, Silverlight and similar technologies are used to play the multimedia items.

Ex.

```
<audio controls>

 <source src="koyal.mp3"
type="audio/mpeg"> Your browser does
not support the html audio tag.
</audio>
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module3-html5/html5/content_type_tag/audio

HTML Video Tag

HTML Video Tag

HTML 5 supports <video> tag also. The HTML video tag is used for streaming video files such as a movie clip, song clip on the web page.

Currently, there are three video formats supported for HTML video tag:

- mp4
- webM
- ogg

Ex.

```
<video controls>
  <source src="movie.mp4" type="video/mp4">
  Your browser does not support the html video
  tag.
</video>
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module3-html5/html5/content_type_tag

Video Attributes

Attribute	Description
controls	It defines the video controls which is displayed with play/pause buttons. height It is used to set the height of the video player.
width	It is used to set the width of the video player.
poster	It specifies the image which is displayed on the screen when the video is not played.
autoplay	It specifies that the video will start playing as soon as it is ready.
loop	It specifies that the video file will start over again, every time when it is completed.
muted	It is used to mute the video output.
preload	It specifies the author view to upload video file when the page loads.
src	It specifies the source URL of the video file.

HTML SVG

What is SVG?

The HTML SVG is an acronym which stands for Scalable Vector Graphics.

HTML SVG is a modularized language which is used to describe graphics in XML. It describe two-dimensional vector and mixed vector/raster graphics in XML.

SVG is mostly used for vector type diagrams like pie charts, 2-Dimensional graphs in an X,Y coordinate system etc.

Ex. Circle

```
<svg width="100" height="100">
 <circle cx="50" cy="50" r="40" stroke="yellow" stroke-width="4" fill="red" />
</svg>
```

cx, cy and r are attributes of circle tag. These attributes can't be used with svg other tag.

Ex. Ellipse

```
<svg height="140" width="500">
  <ellipse cx="200" cy="80" rx="70" ry="50"
style="fill:yellow;stroke:purple;stroke- width:2" />
Sorry, your browser does not support inline SVG.
</svg>
```

The cx attribute defines the x coordinate of the center of the ellipse The cy attribute defines the y coordinate of the center of the ellipse The rx attribute defines the horizontal radius
The ry attribute defines the vertical radius

Ex. Rectangle

```
<svg width="200" height="100">
  <rect width="200" height="100" stroke="yellow" stroke-width="4" fill="red" />
</svg>
```

cx, cy and r are attributes of circle tag. These attributes can't be used with svg other tag.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

HTML SVG Line, Polygon, Polyline

SVG Line

Ex.

```
<svg height="210" width="500">
  <line x1="0" y1="0" x2="200" y2="200" style="stroke:blue;stroke-width:2" />
</svg>
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module3-html5/html5/content_type_tag/SVG

SVG Polygon

The <polygon> element is used to create a graphic that contains at least three sides.

Polygons are made of straight lines, and the shape is "closed" (all the lines connect up).

Ex.

```
<svg height="210" width="500">
  <polygon points="200,10 250,190 160,210"
style="fill:lime;stroke:purple;stroke- width:1" />
</svg>
```

SVG Polyline

The <polyline> element is used to create any shape that consists of only straight lines (that is connected at several points)

Ex.

```
<svg height="200" width="500">
  <polyline points="20,20 40,25 60,40 80,120 120,140 200,180"
  style="fill:none;stroke:black;stroke-width:3" />
</svg>
```

SVG Path, Text and Stroking

SVG Path

The <path> element is used to define a path.

The following commands are available for path data: M =

moveto

L = lineto

H = horizontal lineto

V = vertical lineto

Z = closepath

C = curveto

S = smooth curveto

Ex. Path

Ex.

```
<svg height="210" width="400">
  <path d="M150 0 L75 200 L225 200 Z" />
</svg>
```

SVG Text

The <text> element is used to define a text.

Ex.

```
<svg height="30" width="200">
  <text x="0" y="15" fill="red">I love SVG!</text>
</svg>

<svg height="60" width="200">
  <text x="0" y="15" fill="red" transform="rotate(30 20,40)">I
love SVG</text>
</svg>
```

SVG Stroke

SVG offers a wide range of stroke properties. In this chapter we will look at the following:

- stroke
- stroke-width
- stroke-linecap
- stroke-dasharray

SVG Stroke Ex.

```
<svg height="80" width="300">
  <g fill="none">
    <path stroke="red" d="M5 20 l215 0" />
    <path stroke="black" d="M5 40 l215 0" />
    <path stroke="blue" d="M5 60 l215 0" />
  </g>
</svg>
```

SVG Gradient

SVG Gradient

A gradient is a smooth transition from one color to another. In addition, several color transitions can be applied to the same element.

There are two main types of gradients in SVG:

- Linear
- Radial

Linear Gradient

The <linearGradient> element is used to define a linear gradient.

The <linearGradient> element must be nested within a <defs> tag. The <defs> tag is short for definitions and contains definition of special elements (such as gradients).

Linear gradients can be defined as horizontal, vertical or angular gradients:

Horizontal gradients are created when y1 and y2 are equal and x1 and x2 differ
Vertical gradients are created when x1 and x2 are equal and y1 and y2 differ
Angular gradients are created when x1 and x2 differ and y1 and y2 differ

Ex. Linear Gradient

```
<svg height="150" width="400">
 <defs>
  <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
   <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
   <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
  </linearGradient>
 </defs>
 <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

Radial Gradient

The <radialGradient> element is used to define a radial gradient.

Ex.

```
<svg height="150" width="500">
  <defs>
    <radialGradient id="grad1" cx="50%" cy="50%" r="50%" fx="50%"
    fy="50%">
      <stop offset="0%" style="stop-
      color:rgb(255,255,255); stop-opacity:0" />
      <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

HTML Canvas

HTML Canvas

The HTML canvas element provides HTML a bitmapped surface to work with. It is used to draw graphics on the web page.

The HTML 5 <canvas> tag is used to draw graphics using scripting language like JavaScript.

The <canvas> element is only a container for graphics, you must need a scripting language to draw the graphics. The <canvas> element allows for dynamic and scriptable rendering of 2D shapes and bitmap images.

Create a Canvas

```
<canvas id="myCanvas1" width="300" height="100"
style="border:2px solid;">
Your browser does not support the HTML5 canvas tag.
</canvas>
```

Canvas Tag with Javascript

```
<script>
var c =
document.getElementById("myCanvas"); var
ctx = c.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,200,100);
</script>
```

Line on Canvas

```
<canvas id="myCanvasLine" width="200" height="100"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>
<script>
var c =
document.getElementById("myCanvasLine"); var
cctx = c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
</script>
```

Circle on Canvas

```
<canvas id="myCanvasCircle" width="200" height="100"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>
<script>
var c =
document.getElementById("myCanvasCircle"); var
cctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI)
; ctx.stroke();
```

Text on Canvas

1. Fill Text

```
<canvas id="myCanvasText1" width="300" height="100"
style="border:1px solid #d3d3d3;">
Sorry! Your browser does not support the HTML5 canvas tag.</canvas>
<script>
var c =
document.getElementById("myCanvasText1"); var
cctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello
```

Text on Canvas

2. Stroke Text

```
<canvas id="myCanvasText2" width="300" height="100" style="border:1px solid
#d3d3d3;">
 Sorry!Upgrade your browser. It does not support the HTML5 canvas tag.</canvas>
<script>
var c = document.getElementById("myCanvasText2");
var cctx = c.getContext
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 4) Mernstack-Bootstrap Basic & Advanced

Bootstrap

- Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.
- Bootstrap is completely free to download and use!
- Quickly design and customize responsive mobile-first sites with Bootstrap, the world’s most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.

1) Bootstrap Introduction
2) Bootstrap Getting Started
3) Bootstrap Grid System
4) Bootstrap Fixed Layout
5) Bootstrap Fluid Layout
6) Bootstrap Responsive Layout

- Bootstrap Utilities

1) Bootstrap Typography
2) Bootstrap Tables
3) Bootstrap Lists
4) Bootstrap List Groups
5) Bootstrap Forms
6) Bootstrap Custom Forms
7) Bootstrap Input Groups
8) Bootstrap Buttons
9) Bootstrap Button Groups

- Bootstrap with CSS: Grid System

1) Bootstrap Images
2) Bootstrap Cards
3) Bootstrap Media Objects
4) Bootstrap Icons
5) Bootstrap Navs
6) Bootstrap Navbar
7) Bootstrap Breadcrumbs
8) Bootstrap Pagination
9) Bootstrap Badges
10) Bootstrap Progress Bars
11) Bootstrap Spinners
12) Bootstrap Jumbotron
13) Bootstrap Helper Classes

1) Bootstrap Modals
2) Bootstrap Dropdowns
3) Bootstrap Tabs
4) Bootstrap Tooltips
5) Bootstrap Popovers
6) Bootstrap Alerts
7) Bootstrap Stateful Buttons
8) Bootstrap Accordion
9) Bootstrap Carousel
10) Bootstrap Typeahead
11) Bootstrap ScrollSpy
12) Bootstrap Toasts

Prerequisites for Bootstrap

- HTML (Mandatory)
- CSS (Mandatory)
- JavaScript/jQuery (Moderate)

Bootstrap

Bootstrap Alternatives

1. Skeleton:
Cons – Limited Templates

2. Foundation
Cons – Difficult to modify codes

3. UIKit
Cons – Limited Utility classes as compared to Bootstrap

4. Bulma
Cons – Still in the development phase

5. Pure
Cons – Limited CSS selectors

6. Powertocss
Cons – No longer Maintained

7. Kickstrap
Cons – No longer Maintained

Setup

- **Through CDN:**

https://getbootstrap.com/docs/4.6/getting-started/introduction/

- **Through Installation:**

https://getbootstrap.com/docs/4.6/getting-started/download/

https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/online_CDN

https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/offline_CDN

Bootstrap Grid System

Bootstrap Grid System

- The Bootstrap Grid System allows up to 12 columns across the page. You can use all 12 columns individually or you can groups the columns together to create wider columns.

- Bootstrap Grid System is responsive and the columns are re-arranged automatically according to the screen size.

Bootstrap Grid System

Bootstrap Grid System

There are four classes in Bootstrap Grid System:

- xs (for phones)
- sm (for tablets)
- md (for desktops)
- lg (for larger desktops)

Example

https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/bootstrap-grid

Bootstrap Fixed Layout

Bootstrap Fluid Layout

Bootstrap Utilities

Bootstrap Utilities

- Bootstrap provides some handful helper classes, for faster mobile-friendly development. These can be used for showing and hiding content by device via media query, combined with large, small, and medium devices.

Bootstrap Typography

Bootstrap Typography

Typography provides some utilities to add additional styles to texts.

These utilities are:

1. Text alignment
2. Text transform
3. Font weight and italics

Bootstrap Tables

Bootstrap Tables

- We can create different types of Bootstrap tables by using different classes to style them.

Example

https://github.com/Brijesh1990/tops-website-development/blob/master/module6-bootstrap-basic-advanced/5.6.0BoostrapTable.html

Bootstrap Forms

Bootstrap Forms

- In Bootstrap, there are three types of form layouts:

1. Vertical form (this is default)
2. Horizontal form
3. Inline form

Bootstrap Rules for Form

- There are three standard rules for these 3 form layouts:

1. Always use <form role="form"> (helps improve accessibility for people using screen readers)
2. Wrap labels and form controls in (needed for optimum spacing)
3. Add class .form-control to all textual <input>, <textarea>, and <select> elements

Bootstrap 4 Form

https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/bootstrap-modal

Bootstrap Input Groups

Input Groups

- The .input-group class is a container to enhance an input by adding an icon, text or a button in front or behind the input field as a "help text".
- Use .input-group-prepend to add the help text in front of the input, and .input-group-append to add it behind the input.
- At last, add the .input-group-text class to style the specified help text.

Sizing

- Add the relative form sizing classes to the .input-group itself and contents within will automatically resize—no need for repeating the form control size classes on each element.

Checkboxes and Radios

- Place any checkbox or radio option within an input group’s addon instead of text.

Multiple inputs

- While multiple <input>s are supported visually, validation styles are only available for input groups with a single <input>.

Multiple addons

- Multiple add-ons are supported and can be mixed with checkbox and radio input versions.

Custom forms

- Input groups include support for custom selects and custom file inputs. Browser default versions of these are not supported.

Bootstrap Buttons

Bootstrap Buttons

- There are seven styles to add a button in Bootstrap. Use the following classes to achieve the different button styles:

1. .btn-default
2. .btn-primary
3. .btn-success
4. .btn-info
5. .btn-warning
6. .btn-danger
7. .btn-link

Bootstrap Images

Bootstrap Images

- Bootstrap supports for images. There are three classes in Bootstrap that can be used to apply some simple style to the images.

For Bootstrap 3

1. img-rounded
2. img-circle
3. img-thumbnail

For Bootstrap 4

1. rounded
2. rounded-circle
3. img-thumbnail

Bootstrap Navbar

Navigation Elements

- Bootstrap provides a few different options for styling navigation elements. All of them share the same markup and base class, **.nav**. Bootstrap also provides a helper class, to share markup and states. Swap modifier classes to switch between each style.

- To create a tabbed navigation menu −

1. Start with a basic unordered list with the base class of **.nav**
2. Add class **.nav-tabs**.

Navbar

- The navbar is one of the prominent features of Bootstrap sites.
- Navbars are responsive 'meta' components that serve as navigation headers for your application or site.
- Navbars collapse in mobile views and become horizontal as the available viewport width increases.
- At its core, the navbar includes styling for site names and basic navigation.

Default Navbar

- Add the classes **.navbar, .navbar-default** to the <nav> tag.
- Add **role = "navigation"** to the above element, to help with accessibility.
- Add a header class **.navbar-header** to the element. Include an <a> element with class **navbar-brand**. This will give the text a slightly larger size.
- To add links to the navbar, simply add an unordered list with the classes of **.nav, .navbar-nav**.

Responsive Navbar

- To add responsive features to the navbar, the content that you want to be collapsed needs to be wrapped in a with classes **.collapse, .navbar-collapse**.

- The collapsing nature is tripped by a button that has the class of **.navbar-toggle** and then features two data- elements.

- The first, **data-toggle**, is used to tell the JavaScript what to do with the button, and the second, **data-target**, indicates which element to toggle.

- Then with a class **.icon-bar** create what we like to call the hamburger button. This will toggle the elements that are in the **.nav-collapse**

Navbar Brands

- Adding images to the .navbar-brand will likely always require custom styles or utilities to properly size.

Navbar Forms

- To add form elements inside the navbar, add the .navbar-form class to a form element and add an input(s). Note that we have added a .form-group class to the div container holding the input.

Navbar Text

- Use the .navbar-text class to vertical align any elements inside the navbar that are not links (ensures proper padding and text color).

Fixed Navigation Bar

- The navigation bar can also be fixed at the top or at the bottom of the page.
- A fixed navigation bar stays visible in a fixed position (top or bottom) independent of the page scroll.

Bootstrap Breadcrumbs

Breadcrumb

- Breadcrumbs are a great way to show hierarchy-based information for a site.
- In the case of blogs, breadcrumbs can show the dates of publishing, categories, or tags.
- They indicate the current page's location within a navigational hierarchy.

Bootstrap Pagination

Pagination

- Pagination, an unordered list is handled by Bootstrap like a lot of other interface elements.

Pagination with Icons

- We can use icons in place of Next and Previous text in the patination. Be sure to provide proper screen reader support with aria attributes and the .sr-only utility

Disabled and Active State

- Pagination links are customizable for different circumstances. Use .disabled for links that appear un- clickable and .active to indicate the current page.

Sizing

- Add .pagination-lg or .pagination-sm for additional sizes.

Alignment

- Change the alignment of pagination components with flexbox utilities.

Bootstrap Spinners

Spinner

- Spinner is also called a **loading indicator**. It is used to display/indicate the loading state of our projects. Bootstrap uses a **.spinner** class to create a Spinner.

Bootstrap Contextual Classes

- Contextual classes are used to color table rows (<tr>) or table cells (<td>)

Class/Context	Color	Description
.active	gray	Used to apply the hover color to the table row or table cell
.success	green	Indicates a successful or positive action
.info	cyan	Indicates a neutral informative change or action
.warning	yellow	Indicates a warning that might need attention
.danger	red	Indicates a dangerous or potentially negative action

Bootstrap Layout: Glyphicon

Glyphicon

- Glyphicons are easily understandable icons and symbols. They look great and hence are widely used in web projects.

- Bootstrap provides total 260 Glyphicons from the Glyphicons Halflings set.

- **Note:** Glyphicons are only supported up to Bootstrap version-3: https://getbootstrap.com/docs/3.3/components/

Bootstrap Layout: Dropdowns

Bootstrap Dropdowns

- Dropdown menus are toggleable, contextual menus, used for displaying links in a list format.
- It facilitates users to choose one value from a predefined list.
- You have to wrap dropdown menu within the class .dropdown to create Bootstrap Dropdown.

Bootstrap Dropdown Divider

- The **class .divider** is used to separate links inside the dropdown menu

Bootstrap Dropdown Header

- The **class .dropdown-header** is used to add headers inside the dropdown menu.

Bootstrap Dropdown Disable an item

- Use the **class .disabled** to disable an item in the dropdown menu.

Bootstrap Dropup

- If you want to open the menu upwards instead of downwards, you need to change the element with class="dropup" instead of class="dropdown":

Split Button Dropdowns

- It is used to show dropdowns as split buttons. Here we use all contextual classes.

Bootstrap Layout: Labels

Labels

- Labels are used to provide additional information about something:
- Use the .label class, followed by one of the six contextual classes .label-default, .label-primary, .label-success, .label-info, .label-warning or .label-danger, within a element to create a label:

Bootstrap Layout: Badges

Badges

- Badges scale to match the size of the immediate parent element by using relative font sizing and em units.
- As of v4, badges no longer have focus or hover styles for links

Button

- Badges can be used as part of links or buttons to provide a counter.

Bootstrap Layout: Jumbotron

Jumbotron

- A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.

Fluid Jumbotron

- To make the jumbotron full width, and without rounded corners, add the .jumbotron-fluid modifier class and add a .container or .container-fluid within.

Bootstrap Layout: Page Headers

Page Headers

- A page header is like a section divider.
- The .page-header class adds a horizontal line under the heading (+ adds some extra space around the element):

Bootstrap Layout: Alerts

Alerts

- Alerts are available for any length of text, as well as an optional dismiss button. For proper styling, use one of the eight **required** contextual classes (e.g., .alert-success).

Link Color

- Use the .alert-link utility class to quickly provide matching colored links within any alert.

Additional content

- Alerts can also contain additional HTML elements like headings, paragraphs and dividers.

Dismissing

- Using the alert JavaScript plugin, it’s possible to dismiss any alert inline.

Bootstrap Layout: Progress Bar

Progress Bar

- Progress components are built with two HTML elements, some CSS to set the width, and a few attributes.
- Bootstrap doesn’t use the HTML5 <progress> element, ensuring you can stack progress bars, animate them, and place text labels over them.

Progress Bar

- Use the .progress as a wrapper to indicate the max value of the progress bar.
- Use the inner .progress-bar to indicate the progress so far.
- The .progress-bar requires an inline style, utility class, or custom CSS to set their width.
- The .progress-bar also requires some role and aria attributes to make it accessible.

Labels

- Add labels to your progress bars by placing text within the .progress-bar.

Height

- We only set a height value on the .progress, so if you change that value the inner .progress-bar will automatically resize accordingly.

Backgrounds

- Use background utility classes to change the appearance of individual progress bars.

Multiple Bars

- Include multiple progress bars in a progress component if you need.

Striped

- Add .progress-bar-striped to any .progress-bar to apply a stripe via CSS gradient over the progress bar’s background color.

Animated Stripes

- The striped gradient can also be animated. Add .progress-bar-animated to .progress-bar to animate the stripes right to left via CSS3 animations.

Bootstrap Layout: List Group

List Group

- List groups are a flexible and powerful component for displaying a series of content.

Active Items

- Add .active to a .list-group-item to indicate the current active selection.

Disabled Item

- Add .disabled to a .list-group-item to make it *appear* disabled.

Links and Buttons

- Use <a>s or <button>s to create *actionable* list group items with hover, disabled, and active states by adding .list-group-item-action.

Flush

- Add .list-group-flush to remove some borders and rounded corners to render list group items edge- to-edge in a parent container (e.g., cards).

Numbered

- Add the .list-group-numbered modifier class (and optionally use an <ol> element) to opt into numbered list group items.

Horizontal

- Add .list-group-horizontal to change the layout of list group items from vertical to horizontal across all breakpoints.

Contextual Class

- Use contextual classes to style list items with a stateful background and color.

Badges

- Add badges to any list group item to show unread counts, activity, and more with the help of some utilities.

Bootstrap Layout: Panels

Panels

- A panel in bootstrap is a bordered box with some padding around its content.

Panel Heading

- The .panel-heading class adds a heading to the panel.

Panel Footer

- The .panel-footer class adds a footer to the panel.

Panel Group

- To group many panels together, wrap a with class .panel-group around them.
- The .panel-group class clears the bottom-margin of each panel

Panel with contextual classes

- To color the panel, use contextual classes (.panel-default, .panel-primary, .panel-success, .panel- info, .panel-warning, or .panel-danger)

Bootstrap Layout: Wells

Wells

- The .well class adds a rounded border around an element with a gray background color and some padding

Well Size

- Change the size of the well by adding the .well-sm class for small wells or .well-lg class for large wells

Bootstrap Plugins: Modal

Modal

- The Modal plugin is a dialog box/popup window that is displayed on top of the current page

Static Backdrop

- When backdrop is set to static, the modal will not close when clicking outside it. Click the button below to try it.

Scrolling Long Content

- When modals become too long for the user's viewport or device, they scroll independent of the page itself. Try the demo below to see what we mean.
- You can also create a scrollable modal that allows scroll the modal body by adding .modal- dialog-scrollable to .modal-dialog.

Vertically Centered

- Add .modal-dialog-centered to .modal-dialog to vertically center the modal.

Tooltips

- Tooltips can be placed within modals as needed. When modals are closed, any tooltip within are also automatically dismissed.

Using the Grid

- Utilize the Bootstrap grid system within a modal by nesting .container-fluid within the .modal-body. Then, use the normal grid system classes as you would anywhere else.

Bootstrap Plugins: Scrollspy

Scrollspy

- Automatically update Bootstrap navigation or list group components based on scroll position to indicate which link is currently active in the viewport.

Bootstrap Plugins: Popover

Popover

- The Popover component is similar to tooltips; it is a pop-up box that appears when the user clicks on an element. The difference is that the popover can contain much more content.

Four Directions

- Four options are available: top, right, bottom, and left aligned.

Dismiss on next click

- Use the focus trigger to dismiss popovers on the user’s next click of a different element than the toggle element.

Bootstrap Plugins: Collapse

Collapse

- Collapsibles are useful when you want to hide and show large amount of content.
- The collapse JavaScript plugin is used to show and hide content. Buttons or anchors are used as triggers that are mapped to specific elements you toggle.

Accordion

- Using the card component, you can extend the default collapse behavior to create an accordion. To properly achieve the accordion style, be sure to use .accordion as a wrapper.

Bootstrap Plugins: Carousel

Carousel

- It works with a series of images, text or custom markup.
- It also includes support for previous/next controls and indicators.
- The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript.

Bootstrap – Login Page

https://github.com/Brijesh1990/tops-website- development/tree/master/module6-bootstrap-basic-advanced/bootstrap- modal

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Bootstrap Project

Please find all GitHub links for projects

https://github.com/Brijesh1990/tops-website-development/tree/master/psdtohtml

https://github.com/Brijesh1990/tops- website-development/tree/master/Live- projectexamples

https://github.com/Brijesh1990/tops-website-development/tree/master/psd

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 4) Javascript And Advanced

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

- ✓ Basic Javascript
- ✓ JS Comment
- ✓ JS Variables
- ✓ JS Datatypes
- ✓ JS Operators
- ✓ JS Loops
- ✓ JS Functions
- ✓ JS Objects
- ✓ JS DOM
- ✓ JS BOM
- ✓ JS OOPS
- ✓ JS Exception Handling
- ✓ JS MISC Function
- ✓ JS Events
- ✓ JS Array
- ✓ JS Async
- ✓ ES6

An Introduction to JavaScript

- **What is JavaScript?**
- *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.
- **JavaScript** ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.

An Introduction to JavaScript

Everything in JavaScript Happens inside an execution context

An Introduction to JavaScript

- JavaScript is a Synchronous Single-threaded Language

Execution

```
var n =2;
function square (num) {
    var ans = num * num;
    return ans;
}
var square2 = square(n);
var square4 = square(4);
```

Why is it called JavaScript?

- 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(**European Computer Manufacturer's Association.**)**, 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 and Opera.
- SpiderMonkey – in Firefox.
- …There are other codenames like “Trident” and “Chakra” for different versions of IE, “ChakraCore” for Microsoft Edge, “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 and Opera.

THE EVOLUTION OF JAVASCRIPT

- JavaScript is currently being used in more than 94% of websites. That means millions of web developers are using JavaScript for web application development. The ease of use and widespread adoption of this programming language is exactly why it continues to become more important to web developers.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

THE EVOLUTION OF JAVASCRIPT

- **The Ubiquity of Web Browsers** – Can you think of a time or place where you aren't within one click of an internet browser? Nope. The web is available everywhere and this makes JavaScript more important and more commonplace. Plus, the widespread adoption ensures dynamic content gets displayed the way it was intended.
- **JavaScript Has More Applications** – Thanks to innovations like Node.js and ReactJS, JavaScript is useful for both front and back end web development. By using cross-platform runtime engines to write server-side code, JavaScript has even more applications and increases efficiencies for web developers.
- **Browsers Improved Processing Speeds** – Thanks to improvements in web browsers throughout the years, JavaScript is being processed even faster than ever. This makes it the most practical choice for web developers, because speed is everything.
- **Frameworks like jquery Reduce Learning Times** – Frameworks like jquery make the learning curve much easier for web developers to master JavaScript. Plus, these frameworks make it easier to create highly professional and interactive web applications efficiently.

JS engines

Rendering Engine
+
JavaScript Engine
+
User Interface
=
Trident
+
Chakra
=
Gecko
+
*Monkey
=
Webkit
+
V8
=
+
JavascriptCore

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 the machine language.
- 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?

- 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.
- 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 for the sake of 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 system 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 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.

What CAN’T in-browser JavaScript do?

- 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 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 contain a 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 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.

https://javascript.info
WWW
https://javascript.info
<script>
...
</script>
https://gmail.com

What makes JavaScript unique?

- There are at least *three* great things about JavaScript:
  - Full integration with HTML/CSS.
  - Simple things are done simply.
  - Support 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 also allows 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:
  - CoffeeScript is a “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 own engine that runs in non-browser environments (like mobile apps), but also can be transpiled to JavaScript. Developed by Google.
- There are more. Of course, even if we use one of transpiled languages, we should also know JavaScript to really understand what we’re doing.

Summary

- JavaScript was initially created as a browser-only language, but is now used in many other environments as well.
- Today, JavaScript has a unique position as the most widely-adopted browser language with full integration with HTML/CSS.
- There are many languages that get “transpiled” to JavaScript and provide certain features. It is recommended to take a look at them, at least briefly, after mastering JavaScript.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Pros and Cons of JS

PROS

- Speed
- Simplicity
- Popularity
- Interoperability
- Server Load
- Rich Inerfaces
- Extended Functionality
- Versatility
- Less Overhead

CONS

- Client-side Security
- Browser Support
- Lack of Debugging Facility
- Single Inheritance
- Sluggish Bitwise Function
- Rendering Stopped

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

JavaScript Uses in Web Designing

1. Websites
2. Web Applications
3. Presentations
4. Server Applications
5. Web Servers
6. Games
7. Art
8. Smartwatch Apps
9. Mobile Apps
10. Flying Robots

JavaScript
Applications

DOM
Document Object Model
document
Root element:
<html>
Element:
<head>
Element:
<title>
Text:
"My title"
Element:
<body>
Element:
<h1>
Text:
"A heading"
Element:
<a>
Attribute:
href
Text:
"Link text"

Browser Events

Data Types

Data Types
Primitive Data Type
Non-Primitive Data Type
Number
String
Boolean
Undefined
Null
Object
Array
Function

What is a JavaScript Loop?

- **Entry Controlled Loops**: Any loop where we check the test condition before entering the loop is an entry controlled loop. In these loops, the test condition determines whether the program will enter the loop or not. These include **for, while, etc.**
- **Exit Controlled Loops**: Any loop where we check the test condition after the statements are executed once is an exit controlled loop. In these loops, the test condition determines whether or not the program will exit the loop. This category includes **do…while** loop.

JavaScript Loops

- We repeat the sequence of instructions until the test condition is true. JavaScript provides the following loop statements to achieve this:
  - while statement
  - for statement
  - do..while statement
  - for…in statement
  - for…of statement
  - labeled statement
  - break statement
  - continue statement

While Statement

While

- A **while** statement in JavaScript executes until our boolean condition evaluates to **true**. This loop is an entry controlled loop.
- If the test condition returns **false**, then control passes to the statement just after the loop.
- If the test condition returns **true**, the loop body is executed and the condition is tested again.
- The syntax for a **while** statement is as follows:
- while (boolean condition)
- statement
- For multiple statements, group them with a block statement ({…}).

While

```
<html>
    <body>
    <script>
    var iterator1 = 0; //iteration variable initialized with 0
    while (iterator1 < 5) //testing the condition
    {
        document.write(iterator1 + 1 + ". " + "DataFlair Tutorial for JavaScript</br>");
        iterator1++; //incrementing the value of the variable
    }
    document.write("</br>Loop end");
    </script>
    </body>
</html>
```

For Statement

For

- JavaScript **for loops** are similar to Java and C language **for loops**. It’s an entry controlled loop. It consists of three parts, separated with a semicolon:
- **Initialization:** It initializes the loop with an iteration variable and executes once at the beginning of the loop.
- **Condition:** It specifies a certain condition in the loop that determines whether the loop body will execute or not. If the condition returns true, the code inside the **for** loop will execute. If it returns false, the control moves onto the statement after the loop.
- **Increment/ Decrement:** This section increments or decrements the value of our iteration variable by 1.

do...while Statement

do...while Statement

- This is an exit controlled loop; the loop body executes at least once, even if the condition is not true. This is because the condition is tested when the loop body ends. If the condition returns **true**, the code inside the loop executes again. If it returns **false**, the JavaScript interpreter exits the loop.

```
do
{
//code executed at least once
}while (condition);
```

for…in Statement

For…in Statement

- This is a modified **for loop** that doesn't require the programmer to know the number of iterations the loop needs to perform. It creates a loop iterating over all the enumerable properties of an object. The **keys** store the key of the current property and we access the property's value with it. This means that the loop executes a specific set of statements for each distinct property in a ***JavaScript Object*.** The working of a **for…in** statement in JavaScript is similar to **for-each** loop in Java.

The syntax of a **for…in** statement in JavaScript looks like this:

```
for (keys in objProperties) {
        statements
}
```

- The reason why the developers created the **for…in loop** is to work with user-defined properties in an object. It is better to use a traditional **for loop** over Array elements with numeric indexes. But, for this example, we are using a numeric array so you can understand how the loop works.

For…of Statement

- In the last JavaScript statement, we had to access the array's values with the help of square brackets [].
- We couldn't access them directly and it makes your program more error-prone.
- So the JavaScript developers got us a new way to access these values directly. This makes our code a lot more efficient than before because the values store the property's value, not the key.

A **for…of statement** looks as follows:

```
for (values of objProperties)
{
statements
}
```

Infinite Loop

- One of the most common mistakes programmers make while using loops is creating an infinite loop. This happens when we accidentally add a condition that always returns true. It is very important that you avoid them in your code, but I still want you to practice these loops. Try thinking of some situations where the condition is always true. Let me give you an example:

```
while(1){
//statements
}
```

JavaScript Objects

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the newkeyword)
- Numbers can be objects (if defined with the newkeyword)
- Strings can be objects (if defined with the newkeyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

- A **primitive value** is a value that has no properties or methods. A **primitive data type** is data that has a primitive value.

- JavaScript defines 5 types of primitive data types:

➢ string
➢ number
➢ boolean
➢ null
➢ undefined

Array

- The JavaScript **Array**class is a global object that is used in the construction of arrays; which are high-level, list-like objects.
- Arrays are list-like objects whose prototype has methods to perform traversal and mutation operations. Neither the length of a JavaScript array nor the types of its elements are fixed. Since an array's length can change at any time, and data can be stored at non-contiguous locations in the array, JavaScript arrays are not guaranteed to be dense; this depends on how the programmer chooses to use them. In general, these are convenient characteristics; but if these features are not desirable for your particular use, you might consider using typed arrays.

Common operations

Create an Array

```
let fruits = ['Apple', 'Banana'] console.log(fruits.length) // 2
```

Access an Array item using the index position

- `let first = fruits[0] // Apple`
- `let last = fruits[fruits.length - 1] // Banana`

Array

Loop over an Array

```
fruits.forEach(function(item, index, array) { console.log(item,
    index)
}) // Apple 0 // Banana 1
```

Add an item to the end of an Array

```
let newLength = fruits.push('Orange')
// ["Apple", "Banana", "Orange"]
```

Array

Remove an item from the end of an Array

```
let last = fruits.pop() // remove Orange (from the end) // ["Apple", "Banana"]
```

Remove an item from the beginning of an Array

```
let first = fruits.shift() // remove Apple from the front // ["Banana"]
```

Array

Add an item to the beginning of an Array

```
let newLength = fruits.unshift('Strawberry')
// add to the front // ["Strawberry", "Banana"]
```

Find the index of an item in the Array

```
fruits.push('Mango')
// ["Strawberry", "Banana", "Mango"] let pos =
fruits.indexOf('Banana')
// 1
```

Array

Remove an item by index position

```
let removedItem = fruits.splice(pos, 1)
// this is how to remove an item
// ["Strawberry", "Mango"]
```

Copy an Array

```
let shallowCopy = fruits.slice()
// this is how to make a copy // ["Strawberry", "Mango"]
```

Array

Remove items from an index position

```
let vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot'] console.log(vegetables)
// ["Cabbage", "Turnip", "Radish", "Carrot"]
let pos = 1 let n = 2 let removedItems = vegetables.splice(pos, n)
// this is how to remove items, n defines the number of items to be removed,
// starting at the index position specified by pos and progressing toward the end of array.
console.log(vegetables) // ["Cabbage", "Carrot"] (the original array is changed)
console.log(removedItems) // ["Turnip", "Radish"]
```

Array

Accessing array elements

```
let arr = ['this is the first element', 'this is the second element', 'this is the last element']
console.log(arr[0]) // logs 'this is the first element'
console.log(arr[1]) // logs 'this is the second element'
console.log(arr[arr.length - 1]) // logs 'this is the last element'
```

Array

Relationship between length and numerical properties

- Several of the built-in array methods (e.g., join(), slice(), indexOf(), etc.) take into account the value of an array's lengthproperty when they're called.
- const fruits = [] fruits.push('banana', 'apple', 'peach') console.log(fruits.length) // 3
- fruits[5] = 'mango' console.log(fruits[5]) // 'mango' console.log(Object.keys(fruits)) // ['0', '1', '2', '5'] console.log(fruits.length) // 6

Array

Increasing the length.

```
fruits.length = 10
console.log(fruits)
// ['banana', 'apple', 'peach', empty x 2, 'mango', empty x 4]
console.log(Object.keys(fruits)) // ['0', '1', '2', '5'] console.log(fruits.length)
// 10 console.log(fruits[8])
// undefined
```

Decreasing the lengthproperty does, however, delete elements.

```
fruits.length = 2
console.log(Object.keys(fruits)) // ['0', '1']
console.log(fruits.length) // 2
```

Array() constructor

The **Array()**constructor is used to create Arrayobjects

```
// literal constructor
[element0, element1, ..., elementN]
// construct from elements
new Array(element0, element1, ..., elementN)
// construct from array length new
Array(arrayLength)
```

Array literal notation

```
let fruits = ['Apple', 'Banana'];
console.log(fruits.length); // 2
console.log(fruits[0]); // "Apple“
```

Array constructor with a single parameter

```
let fruits = new Array(2); console.log(fruits.length); // 2
console.log(fruits[0]); // undefined
```

Array constructor with multiple parameters

```
let fruits = new Array('Apple', 'Banana');
console.log(fruits.length); // 2  console.log(fruits[0]);
// "Apple"
```

Array.from()

The **Array.from()**static method creates a new, shallow-copied Arrayinstance from an array-like or iterable object.

```
console.log(Array.from('foo'));
// expected output: Array ["f", "o", "o"]
console.log(Array.from([1, 2, 3], x => x + x));
// expected output: Array [2, 4, 6]
```

Parameters

arrayLike

An array-like or iterable object to convert to an array.

mapFn Optional

Map function to call on every element of the array.

thisArg Optional

Value to use as this when executing mapFn.

Return value

A new Array instance.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects. There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using Object.create().

Using an Object Literal

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs (like age:50) inside curly braces {}.
- The following example creates a new JavaScript object with four properties:
- const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Using the JavaScript Keyword new

- const person = new Object();
  person.firstName = "John";
  person.lastName = "Doe";
  person.age = 50;
  person.eyeColor = "blue";

JavaScript Properties

- Accessing JavaScript Properties

  The syntax for accessing the property of an object is:

- *objectName.property* // person.age

  or

- *objectName*["*property*"] // person["age"]

  or

- *objectName*[*expression*] // x = "age"; person[x]
- The expression must evaluate to a property name.

JavaScript Object Constructors

- function Person(first, last, age, eye)
  { this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
  }

Object Types (Blueprints) (Classes)

The examples from the previous chapters are limited. They only create single objects.
Sometimes we need a "**blueprint**" for creating many objects of the same "type".
The way to create an "object type", is to use an **object constructor function**. In the example above, function Person()is an object constructor function.
Objects of the same type are created by calling the constructor function with the new keyword:

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

Object Prototpye

- JavaScript is a dynamic language. You can attach new properties to an object at any time as shown below.

```
function Student() { this.name = 'John'; this.gender = 'Male'; } var studObj1 = new
Student();
studObj1.age = 15; alert(studObj1.age); // 15
var studObj2 = new Student();
alert(studObj2.age); // undefined
```

As you can see in the above example, age property is attached to studObj1 instance. However, studObj2 instance will not have age property because it is defined only on studObj1 instance.

So what to do if we want to add new properties at later stage to a function which will be shared across all the instances?

The answer is **Prototype**.

Object Prototpye

- The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.

Object Prototype

```
function Student()
      {  this.name = 'John';
      this.gender = 'M';
}
var studObj = new Student();
console.log(Student.prototype);                    //    object
console.log(studObj.prototype);                    //    undefined
console.log(studObj.__proto__);                    //    object
console.log(typeof Student.prototype); // object  console.log(typeof studObj.
proto_); // object  console.log(Student.prototype === studObj.__proto__); // true
```

Object's Prototype

- As mentioned before, object's prototype property is invisible.
  Use Object.getPrototypeOf(obj)method instead of_____proto____to access prototype object.

```
function Student() {
        this.name = 'John';
        this.gender = 'M';
}
var studObj = new Student();  Student.prototype.sayHi=
function(){ alert("Hi"); }; var studObj1 = new Student();
var proto = Object.getPrototypeOf(studObj1); // returns Student's prototype object alert(proto.constructor); //
returns Student function
```

Prototype object properties and methods.

Property	Description
constructor	Returns a function that created instance.
__proto___	This is invisible property of an object. It returns prototype object of a function to which it links to.

Prototype object properties and methods.

Method	Description
hasOwnProperty()	Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.
isPrototypeOf()	Returns a boolean indication whether the specified object is in the prototype chain of the object this method is called upon.
propertyIsEnumerable ()	Returns a boolean that indicates whether the specified property is enumerable or not.
toLocaleString()	Returns string in local format.
toString()	Returns string.
valueOf	Returns the primitive value of the specified object.

Changing Prototype

```
function Student()
    {  this.name = 'John';
    this.gender = 'M';
}
Student.prototype.age = 15;  var
studObj1 = new Student();
alert('studObj1.age = ' + studObj1.age);                    // 15
var studObj2 = new Student();
alert('studObj2.age = ' + studObj2.age);                    // 15
Student.prototype = { age : 20 };
var studObj3 = new Student();
        alert('studObj3.age = '  +      studObj3.age);   // 20
        alert('studObj1.age = '  +      studObj1.age);   // 15
        alert('studObj2.age = '  +      studObj2.age);   // 15
```

Inheritance in JavaScript

- Inheritance is an important concept in object oriented programming. In the classical inheritance, methods from base class get copied into derived class.
- In JavaScript, inheritance is supported by using prototype object. Some people call it "Prototypal Inheriatance" and some people call it "Behaviour Delegation".
- Let's see how we can achieve inheritance like functionality in JavaScript using prototype object.
- Let's start with the Person class which includes FirstName & LastName property as shown below.

Inheritance in JavaScript

```
function Person(firstName, lastName) {
        this.FirstName = firstName || "unknown"; this.LastName = lastName || "unknown";
};
Person.prototype.getFullName = function () {
        return this.FirstName + " " + this.LastName;
}
```

Inheritance in JavaScript Multiple

```
function Person(firstName, lastName) {
        this.FirstName = firstName || "unknown";
        this.LastName = lastName || "unknown"; }
        Person.prototype.getFullName = function () {
                return this.FirstName + " " + this.LastName;
        }
function Student(firstName, lastName, schoolName, grade) {
        Person.call(this, firstName, lastName);
        this.SchoolName = schoolName || "unknown";
        this.Grade = grade || 0; }
        Student.prototype = new Person();
        Student.prototype.constructor = Student;
        var std = new Student("James","Bond", "XYZ", 10);
        alert(std.getFullName()); // James Bond
        alert(std instanceof Student); // true alert(std instanceof Person); // true }
```

JavaScript Closure

- Closure is one of important concept in JavaScript. It is widely discussed and still confused concept. Let's understand what the closure is.
- First of all, let's see the definition of the Closure given by Douglas Crockford: crockford.com/javascript/private.html
- *Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.*
- You have learned that we can create nested functions in JavaScript. Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function). Consider the following example.

```
function OuterFunction() {  var
            outerVariable = 1;
      function InnerFunction()
            { alert(outerVariable);
      }
      return InnerFunction;
}
var innerFunc = OuterFunction();
innerFunc(); // 100
```

Use Of *Closure*

```
function Counter() {
        var counter = 0;
        function IncreaseCounter() {
            return counter += 1;
        };
        return IncreaseCounter;
}
var counter = Counter();
        alert(counter());   // 1
        alert(counter());   // 2
        alert(counter());   // 3
        alert(counter());   // 4
```

Closure

```
function Counter() {
var counter = 0; setTimeout( function () { var innerCounter = 0;
counter += 1;
alert("counter = " + counter);
setTimeout( function () {  counter += 1;
innerCounter += 1;
alert("counter = " + counter + ", innerCounter = " + innerCounter) }, 500);
}, 1000);
};
Counter();
```

When to use Closure?

- Closure is useful in hiding implementation detail in JavaScript. In other words, it can be useful to create private variables or functions.

```
var counter = (function() { var privateCounter = 0; function changeBy(val) {
privateCounter += val; } return {
increment: function() {
            changeBy(1);
},
decrement: function()
{ changeBy(-1);
}, value: function() {
return privateCounter;
} }; })();
alert(counter.value());          // 0
counter.increment();
counter.increment();
alert(counter.value());          // 2
counter.decrement();
alert(counter.value());          // 1
```

Iterators and Generators

What is Iterables and Iterators?

ES6 introduces a new mechanism for traversing data: *iteration*. Two concepts are central to iteration:

- An *iterable* is a data structure that wants to make its elements accessible to the public. It does so by implementing a method whose key is Symbol.iterator. That method is a factory for *iterators*.
- An *iterator* is a pointer for traversing the elements of a data structure (think cursors in databases)

Iterable values in Javascript

- The following values are iterable:
  - Arrays
  - Strings
  - Maps
  - Sets
  - DOM data structures (work in progress)

The idea of iterability is as follows.

- **Data consumers:** JavaScript has language constructs that consume data. For example, for-of loops over values and the spread operator (...) inserts values into Arrays or function calls.
- **Data sources:** The data consumers could get their values from a variety of sources. For example, you may want to iterate over the elements of an Array, the key-value entries in a Map or the characters of a string.

The following ES6 language constructs make use of the Iterable:

Destructuring via an Array pattern
for-of loop
Array.from()
Spread operator (...)
Constructors of Maps and Sets
Promise.all(), Promise.race()
yield*

What are Generators?

- You can think of generators as processes (pieces of code) that you can pause and resume while executing that particular code :

```
function* func() {
// (A)
console.log('First');
yield;
console.log('Second');
}
```

The yield keyword pauses generator function execution and the value of the expression following the yield keyword is returned to the generator's caller. It can be thought of as a generator-based version of the return keyword.

Generator function declarations:

- function* genFunc() { … }
  const genObj = genFunc();

Generator function expressions:

- const genFunc = function* () { ⋯ };
  const genObj = genFunc();

Generator method definitions in object literals:

- const obj = {
  * generatorMethod() {
  …
  }
  };
  const genObj = obj.generatorMethod();

Generator method definitions in class definitions

- class MyClass {
  * generatorMethod() {
  …
  }
  }
  const myInst = new MyClass();
  const genObj = myInst.generatorMethod();

Usecase of Generators:

Simpler asynchronous code

we can execute simple asynchronous block of code.

Receiving asynchronous data using generators

ES8 will have async functions(async/await) which are internally based on generators. With them, the code looks like this:

```
async function fetchJson(url) {
    try {
        let request = await fetch(url);
        let text = await request.text();
        return JSON.parse(text);
    }
    catch (error) {
        console.log(`ERROR: ${error.stack}`);
    }
}
```

Implementing Iterables

- function* objectEntries(obj) {

```
        const propKeys = Reflect.ownKeys(obj);

        for (const propKey of propKeys) {
        // `yield` returns a value and then pauses
        // the generator. Later, execution continues
        // where it was previously paused.
        yield [propKey, obj[propKey]];
        }
}
```

- The above function returns an iterable over the properties of an object, one [key, value] pair per property, The objects returned by generators are iterable,using that iterable we can get next yield of [key,value] pair use of *for-of* loop,

The following code use that Object.entries() method

- const user= { name: 'raja', age: 25};
  for (const [key,value] of objectEntries(user))
  { console.log(`${key}: ${value}`);
  }
  // Output:
  // name: raja
  // age: 25

Example : https://javascript.info/first-steps

ES6 : default function parameter

- if we don’t pass argument to function then it take that argument undefined

-so if we set default function parameter then function will execute with that default value if don’t pass the value

Example

ES6 : Promises

Make easy to work with asynchronous call

If we write

reject('Things went wrong')

response(['7,4,1'])

No App will crash only reject call then no other code execute

Use Promise with MongoDB

Mongo has in built support for Promises

Example

TOPSTECHNOLOGIES

Training | Outsourcing | Placement | Study Abroad

Module – 6

[React JS]

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Modules

Module-1) React - Components, State, Props

Module 2) React – List, Hooks, Local storage,Api

Module 3) React – Advance React Styling , Routing

Module 4) React – JSON-server and Firebase Real Time Database

Module-5) React - Applying Redux

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module - 1

[React - Components,State,Props]

What is React?

- The Javascript library for building user interfaces.
- Used to build highly **interactive**, **flexible**, **responsive** web application.
- Traditional web applications were not this fast and rapid.
- ReactJS is the client side javascript library.

Single page application

- React is used to build **SPA**

DOM

- DOM- Document object model

DOM

- DOM- The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

Why React?

- Javascript can help in developing the reactive web application but it is often found that javascript needs lot of code to do simple actions as well.
- Creating element, adding class or styles,appending it to the element.
- Have to add event listeners to the elements.

Why React?

- Using react you can simple out this steps and can split your logic in multiple components.
- All the low level instructions are handled by react library itself.
- You can define the reusable components.

React vs Angular vs Vue

- All three are very popular and highly used frameworks.

React	Angular	Vue JS
Component based	Component based	Component based
Very focused features	Many more built in features	Medium built in features
Javascript	Typescript	Javascript

About the course

- Basic features
  - Key concepts
  - Components and screens
  - Handling events and passing data
  - Adding styles to the pages
- Advanced features
  - Routing
  - State management
  - HTTP requests
  - Forms and authentication

NOTE:

For each concept we will have small demonstrations to actually see how the implementations.
And we will have major project to see integrated approach and how to use all concepts together.

Getting most of the concepts

- Go through the videos one by one
- Practice along with the modules.
- Try to solve error and bugs at your own, finding the root cause.
- Complete all the assignments given at module level.
- Attempt all the quiz and if not answered correctly refresh the concept.
- For more detailed information you can always refer the official documentation for react.

Setting up the environment

- Code Editor - Microsoft visual code.

Visual Studio Code

- There are some by default extensions already added. Add one more extension named **Prettier** for code formatting.
- key shortcut used to bind any shortcut for formatting the code

React Components

Basics of React

- Looking into all the basic features that react provide us as library.
- How to build the interactive user interface and user friendly UIs.
- What are components and how to use them as core building blocks.
- Knowing about RSX.
- How to pass data from one component to another component.
- Building the project through the course to know how to use all the theoretical concepts in real time application

Features

- Now we know that we can use javascript, html and css for building web applications.
- But React is used to make this process simpler with simpler code and simpler structure.

Components

- React is all about components.
- It is the combination of html, css and javascript

Why Components?

- ❑ Reusability
- ❑ Single responsibility

Components

- How to write up code in components?
- React allows you to create reusable and reactive sections consisting HTML and Javascript with css.

Components

- React uses approach called **declarative approach.**
- Just maintaining states to be achieved under certain condition.

Installing NodeJS

- There is the tool called **create react app** which can be used to create the project with default configurations and setup
- Creates the development server to run the application locally.
- NodeJS is used to run javascript without using browser and here used by create react app tool
- Simply download and install the latest version of nodejs

Creating the first project

- Having nodejs one can execute the command to create your project with the help of the tool we talked about.
- *npx create-react-app my-first-app*
- The command to create a new project is:
- Go to that folder and execute the following command to run the project

  npm start

Files structure

- Open the folder in code editor
- There you will see some default files already created
- This might change with time, but you will have src folder where your code exists
- Package.json to maintain all the dependencies.

Files structure

- To install all the dependencies you have the following command to be

  npm install

- This will create the folder named **node_modules** which will have all the files installed.

NOTE:

No need to run this as npx create-react-app already executes npm install

Understanding index.js

- Npm start will transform and optimize the code into something that browser can understand.
- The first file that gets used when you open the application is index.js
- Ultimately the code we write gets converted into javascript
- You can define custom tags that looks similar to html tags, e.g <App />

Understanding index.js

- Index.html in public folder is the only html file that gets rendered in the browser as we are dealing with Single page application.
- The imported App is the component.
- In modern js if you want to use any function or class outside the file,you have to export and import it.

Introducing JSX

- JSX stands for javascript xml.
- The html code is not valid in javascript, but here in app.js we have some code because react has introduced JSX.
- The code is not understood by browser, but react transforms the code in pure javascript
- Pure javascript follows the imperative approach, where you give clear instructions where to render, what to render.
- We have our first component App in the project.

Custom Component

- Create our first custom component.
- Add folder called components which will have all the components.
- App component will not be included in this as it is special kind of component kinda root.
- Create file named Subscription.js

Custom Component

- Components are just function with some special syntax, so here we will define the function.
- To use this component we have to export the function.
- We will import this in App.js and will use the identifier in html code.
- The custom tags should have upper case first letter and all the smaller case are treated as default tags.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

React Introduction to JSX

Adding template

- Let’s try to add some more elements in the Subscription component.
- There is one thumb rule for JSX code, it should have only one root element.
- The easiest way to solve the problem is by wrapping the template with one more div element.
- Can also add parentheses to make it more readable.

Adding styles

- For this we will add one more file named Subscription.css under components folder.
- You have bind this css file with your .js file for that component.

```
Import ‘./Subscription.css’
```

- To add class we will use className because the code is JSX and not actual html.

Dynamic Data

- Show multiple subscriptions and render the data dynamically on the list.
- We know components add reusability in the application.
- And we do not want to have hardcoded values. So here we will add variable for date, title and amount.
- We will use this variables in JSX with single '{}'.

  {date} or {title} or {amount}

- As date is not of type string we have to change it to string by using .toISOString() method of vanilla javascript.

Passing Data

- To show multiple subscription with different data, we will reuse the same component multiple times in app.
- Pass the data from parent component by using attributes.
- Here use the concept of props instead of attributes and get the access from the custom components.
- Add the array of subscriptions in the app component of object type having three fields: date, title and amount.
- Pass the attributes to the subscription element.

Accessing Props

- Once you pass the values from parent, it is necessary to access those values in the component.
- To get the access in the subscription component we will use parameters.

```
function Subscription(props){}
```

- It is ensured by React that props contains all the passed attributes.
- Now when I want to use any attribute I will write:

```
{props.title} or {props.date}
```

JS code in JSX

- For date we want to show more readable format.
- To show date in certain format we can simple use javascript functions and logic and transform it.
- To keep the code simple and readable, we can maintain variable or const and just use those in the html snippet.

Single responsibility

- Splitting the application in smaller building blocks.
- Here we can separate out few things into another component.
- New Component for rendering date.
- Shift all the logic and template for date to this component and export the function.

Single responsibility

- Use this component in the subscription component for date.
- We will pass the date from parent component using props to this new component.
- For adding styles lets add .css file and import it in .js file.

Children Props

- Still we have same kind of template at multiple places.
- Add one more component - Container and .css file
- Use this container in Subscription.js instead of div.
- You cannot use custom elements as wrapping element in JSX.
- But react gives the special handling for such cases, where you use **props.children**
- To pass some classes to be used you have to add logic for handling classes in container class.

JSX in detail

- Look at the code actually loaded in browser from developers tools.
- You can replace your code in app.js with the below lines.

```
return React.createElement('div',{},React.createElement('h2',{},
  'Let's start!'))
```

- So you need to import react in past as ultimately this was the code.
- Because react works like this you need to have one element as root element, otherwise you cannot just have this kind of object returned.

Folder structure

- So far we have kept all the components at single level.
- But when the application grows and we have more and more components we cannot keep files in such way.
- Categories the components based on its role and put them under different folders.
- Have to update imports according to new hierarchy.

Arrow function

Using arrow function syntax:

- Arrow function is not related to react at all, it is just javascript.

*const App = () => {*ow function replacing the keyword function.

- Lets update all the components with arrow function.

```
Arrow Functions

() => {
  ...
}
```

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

React

Events and state

Event Listening

- Listen to the user actions and handle those events to make application reactive and not stick to the static state.
- Add button in Subscription component in the end to change title and bind logic on click event
- For that you can use imperative approach like

```
document.getElementById('root').addEventListener()
```

- But will use JSX syntax by adding onClick.
- Can have inline function or inline arrow function.

Event Handling

- To handle the user click on the added button, define the arrow function which will be binding with onClick event
- Can also define the function with "function" keyword.
- But then you have to add parentheses which will execute the function when the template is evaluated.

Event Handling

- So we will write up arrow function and not normal function for event handling.
- For changing title we will maintain variable and will update that in the method binding.
- Although we update the variable, it will not reflect on the screen

State

- When you update the value in function, you have to inform react to evaluate DOM.
  Import react, { useState } from 'react'
- Call useState() function from component function with the default value of props.title.
- useState not just create special variable but provides the function to update that variable.

```
const [title,setTitle]=useState(props.title)
```

Deep dive in state

- State is the key feature provided by React.
- useState maintains the variable for each instance of component.
- Why using const for useState when updating value of title.
- But here not assigning any value, just calling setTitle() method for update.
- One get latest value of special variable in title which we used in array.

Forms in React

- Add new folder under components folder named NewSubscription and define new component.
- Define another component which will have all the user inputs.
- Also define the .css file for FormSubscription component.
- There should be three inputs
    - Title
    - Date
    - Amount

Using created form

- Let’s import the form in the NewSubscription component.
- Add the template having the custom tag of subscription form.
- Add the .css file for this subscription component.
- Ultimately we want to use this NewSubscription component in the app component.
- Import NewSubscription component and use that in the template.

Fetching user inputs

- Add handlers for every input.
- Listen to onChange event for each input and bind it with the logic.
- Can define inline arrow function here but we will add functions on top.
- In vanilla JS we get event object by default when we listen to events, similarly we will get event as argument by default
- To get the value we can use **event.target.value**

Storing the inputs

- For that again, we will use state and define the array [userTitle,setTitle]
- And in the handler we will call setTitle function with the value as parameter.
- Add event listener for all three inputs.
- Add useState for all three inputs.
- A component can have **multiple states** maintained

Another approach for storing

- Defined three states for three fields and that is okay.
- But here we can use single state for all three as they are the part of single form

  useState({userTitle:”,userDate:”,userAmount:”})

- When you update the value of this state you have to update all three fields, because it overwrites the old version and will not merge it.

  setForm({...Form,event.target.value})

Better approach

- Updating the value of state that depends on the previous state is not a good practice.
- Passing the arrow function in the setForm method.

```
setForm(()=>{})
```

- By default it will get the parameter for previous state and then we will return the new state.

```
setForm((prevState)=>{ return{...prevState,userTitle:event.target.value}})
```

- React schedules the state updates and doesn’t updates instantly.

Submitting the form

- Add onClick event listener for button, but when the button is of type submit we should not do that.
- Instead we will use onSubmit method with the form tag.
- Define handler the on top and use that function to be binded.

Submitting the form

- Use preventDefault()
- When we are using single state we already have the object but when we have multiple states, we need to combine them in one object.
- Still for single state we might need to reassign the values as we need different data types

Two-way binding

- We are using state to store the user inputs.
- States can also be used for two way binding
- We can use the userTitle of the state as the value of the first input.
- Similarly add value attributes for all three inputs.
- Once the form gets submitted we will set the states to the empty string, as we want to reset the form.

Component communication

- Need the data gathered in the app component because we have the array in the app.
- We know using props we can send the data from parent to child.
- The way we use onChange in input element where we pass some function from parent to child.
- Pass the data from form component to newSubscription and then to app component.
- Add the prop for form tag and define the handler method.

Using the handler

- On adding in custom component have to handle this step.
- In Submit handler execute the function passed as pointer using props.
- Now having the data in newSubscription pass it to app in the same way using props.
- Define the prop for newSubscription and handler in app component.
- Execute the function inside the onSaveHandler.

Lifting up the state

- Using one section for generating the data or state i.e. NewSubscription.
- Cannot just pass the data between sibling components.
- Pass data to App and from App to Subscription.
- This concept is called lifting the state up from newSubscription to App.
- And we are passing the lifted state to child component by using props.

Adding filter component

- Enhance the project by adding one more component used to filter out the data based on the user selection.
- Add .js and .css file for Filter component.
- Add the template for the component and add styling in the css file.

Adding filter component

- Use select which is built in component and listen to onChange prop.
- In app component use this newly created component inside of container.
- Add the prop onFilterChange and define the handler.
- Call onFilterChange using props in the handler in filter component.

Storing the value

- The next step is to store the value.
- Use state and define a new state in parent component.
- From the filterchanged handler update the value of the state **selectedFilter**
- Can add two way binding by passing the state to filter component.
- Inside the filter component set the value of select component to **props.selectedFilter**

Controlled vs Uncontrolled

- The filter component do not have any logic or any data to be maintained.
- One use props for getting the data and for passing the data
- Such components are called **controlled** component.
- Components that are only used for rendering few elements.
- Many terms are used for such concept
    - Presentational **Vs** stateful component.
    - Stateless **Vs** stateful component
    - Dumb **Vs** smart component

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

React

Iterations and conditions

Iterating list in JSX

- Dynamic listing of the items, to reduce the repetitive code.
- And also adding some condition based logic which should be used to render the content
- Added the Subscription tag multiple times to render each list item.
- All the javascript execution is done using {} in the JSX
- Use .map js function to transform each object into JSX element.

```
{props.item.map(subscription => <Subscription />)}
```

Array with state

- Already have the newly added subscription in the app component.
- Add that in the original array.
- Use state to maintain the array and update the data.
- During the handler update the array using set method.
- Use the better way to update, pass arrow function as argument of set method.

Unique key

- React handles the iteration rendering over list in its own way.
- When we add element, it will just add one node at last and will update all the elements to match the order.
- Beside performance issue, we can also have bugs.
- To get rid of this, we can add prop called **key** in Subscription tag which will not be used by component

```
key ={element.id}
```

Filtering the data

- But still our filters are not working.
- Based on the filter applied I want to modify the list.
- In app component, we have the selected filter and we have the list of objects.

Filtering the data

- We can simply use .filter() method of array

```
const  filteredData = props.items.filter(item=>{return
                                item.date.getFullYear().toString === filteredYear})
```

- Use this new filteredData to be iterated in JSX

Conditions

- Conditional rendering: to render template 1 in one condition and template 2 in other.
- Here when no item in the filteredData, inform user the same
- For that we cannot use if statements in {} because long statements are not allowed but we can use ternary operator.

```
FilteredData.length === 0? <p>No data found.</p> : {{map function}}
```

Conditions using &&

- Can remove ternary operator by using && operator.

*FilteredData.length ===0 && No data found. *

- If the first conditions is true, it will return second statement.
- Update the below statement to render the list.
- This is just the alternative that can be used to get rid of complex ternary operator.

Conditions using &&

- Define the variables to hold the content used in JSX code outside of return block.

  let content= No data found
- Updating the content based on the condition using simple javascript if statement.
- And using this variable directly in the JSX
- This is also the alternative to add conditions

Conditional return

- The logic for filter in the same component, is not recommended.
- Add new component -- subscriptionList and shift all the logic in that component and add the css file.
- In place of the variable, use the tag for newly created component and will pass props filteredData.
- Return the JSX based on the condition.

Adding handlers

- Do not show the form every time, but instead keep a button to open that form.
- New state to decide whether to show form or to show button
- Add the click event and bind it to showFormHandler.
- Use && operator with showForm to show or hide the content based on the state.

Adding handlers

- Can see the button and on clicking the button can see the form.
- Close the form on submit or on cancel.
- In newSubscription component add new function to hideForm.
- And pass this function to the subscriptionForm component to be executed when clicked on cancel.

  onCancel={hideForm}
- Update the state to false in the save function in newSubscription.

Global CSS

- Till now though we have separate css files for each component.
- These css classes are global.
- If we have defined any class to other file, it will be still applied.
- Not defined any scope for the classes we have defined.
- In next module will see this concept and alongwith that other styling concepts as well.

Module - 4

[Lists and Hooks]

Limitations of JSX

- In react we are using JSX for rendering our components and creating templates.
- But we have some limitations in JSX, we cannot have multiple elements.
- We know the solution, wrapping element.
- We can use the array and remove the wrapping element, and add commas between dynamic values and elements.
- But when react works on the array of jsx elements, it requires key.
- You can provide the hard coded key.
- But this is not the solution we follow, having wrapping element is way easier.

Wrapper component

- We have the solution to the JSX limitation.
- Container component is already defined which is used as wrapper component.
- But we can add another component which will not have any JSX code.
- Create one component named wrapper in UI folder.
- Just return **props.children** from the main function.
- And now just use this wrapper tag in addSubscriber instead of div.

React fragment

- React provides you one way to add this wrapper component.
- **React.Fragment** is used as the wrapping element.
- You can just define your template under empty tags.

```
<> <h2>Title</h2></>
```

- This will not be valid for all projects you need some special configurations.
- If you do not like this or if not valid in your project you can use

```
<React.Fragment><h2>Title</h2></React.Fragment>
```

- You can also add fragment in your import and can directly use it.

```
Import {useState,Fragment} from 'react'
```

React portals

- Let add two div as root element in the index.html above element with id root.
   and
- Add one more component in errorModal as we are just going to use it here and copy the div with class backdrop to be returned.
- Similarly add Modal component and copy the container section.

React portals

- We will use react-dom which comes with react
  <React.fragment> ReactDom.createPortal() </React.fragment>
- With two arguments
  - <Backdrop onClose=props.onClose></Backdrop>
  - document.getElementById("backdrop-root")

React portals

- Just as we used portal to add the backdrop element we will use portal for overlays.
- Repeat the same steps by adding ReactDom.createPortal() with two arguments
    - <Modal title={props.title}content={props.content} onClose={props.onClose}>
    - document.getElementById("overlay-root")

React Refs

- Maintain states for storing the user input and updates the state on every keystroke.
- To improve this behavior we have another react hook useRef.
- It is used to connect the html elements with javascript and it takes default value just as in useState.

```
const nameInputRef= useRef()
```

React Refs

- Another props that is **ref** which is similar to key which is used by react.
- And in ref we will assign the respective const.
- In addSubscriber handler we will console the value of refs
- And we can just replace the state in handler with refs.
- For resetting we have to change the value of input. nameInputName.current.value=”
- Always try to avoid the DOM manipulation, here we are just updating the value.

Uncontrolled inputs

- By using the ref we are making the input components uncontrolled
- We are resetting the value by directly accessing the dom element and using the dom api, we are not using react here.
- The input components maintain its internal state and reflects it.
- When we use states the components are controlled, as in every keystroke we update the state and we pass the data back using value.

Effects

- Now in this module we will see what are effects and working with it.
- We will see how to manage complex state with reducers.
- What is context and how to use them to manage app-wide state.
- Side effects is anything else happening in the application.
- For example http requests, storing something in local storage, etc. which is not updating the DOM directly.

Effects

- That means those activities which are not directly used to render something must be outside the main function.
- So to handle such side effects we have another react hook that is useEffect()
- useEffects takes two arguments
  - Arrow function - runs when the dependencies specified changes.
  - Dependencies - the list of things which you want to handle.

Localstorage

- In the first project we have provided the default filtered year, but if the user changes the option and reloads the application the selected year gets lost.
  localstorage.setItem('filteredYear',selectedYear)
- When the page gets reloaded in the same component,check localstorage
  localstorage.getItem('filteredYear')
- If anything found update the state maintained for it.

Using Effect

- Updating the state, it will go into infinite loop.
- Because when the state is updated, the components gets evaluated again.
- For that import useEffect from react and shift the condition in the first argument's arrow function.
- And in the second argument pass empty array, so that by default it will be executed only once when the component gets initialized, then no dependencies will be updated to call it again.

Adding dependencies

- Now we can use effect in form and shift the code to set the error state and pass empty array of dependencies.
- But here only once the state will be updated and that is not the case required.
- If we remove the dependencies it will be executed everytime the component gets evaluated, which will turn into infinite loop.

Adding dependencies

- Define the dependencies, on change of which we want to run the function of updating the error state.
- Now it might be confusing that here we do not have any timer or http request, but actually we are updating the state.
- Then why to use effect
- But we are storing user input on every keystroke and side by side we are checking the validity.

What to add or not add

- Should we add everything as dependencies which we are using in the function in useEffect()
- We should only add those dependencies that will change based on user input or parent component re-rendering.
- State updating function, built-in functions of javascript
- Not to add those variables or functions that are defined outside of component function

Adding delays

- Sometimes you need to cleanup things.
- On every keystroke our validation will be checked, here the function in not complex, but imagine you are calling api for this.
- Using debouncing we can wait for user to complete typing.
- SetTimeout for 500 ms to delay the process.
- On every keystroke we will clear all past timers and will use only latest.

Clean up function

- Using effects makes it easy to clear all past timers and maintain only one timer at a time.
- Do this by using cleanup function.
- For that return anonymous arrow function
  return ()=>{}

- So except for the first time, whenever the side effect function will run, it will first run clean up function and also when components gets removed

clearTimeout()

- Settimeout functions returns some identifier so we will store that in const.
- Default function called clearTimeout() which is used to remove the scheduled timer.
- Use this function in the clean up function by passing the const used to store the identifier.

Reducer

- It is simple to use state when we have simple data to be managed.
- Sometimes you have multiple states, multiple ways to update it or dependencies to other state.
- Reducer is the replacement of state for more powerful state management.
- We can also use reducers when we have state update dependent on any other state.
- Example: form validity state is dependent on the userTitle.

useReducer()

- Just like useState we have to import useReducer from react.
- UseReducers also returns two values:
  - State
  - Dispatch Function to trigger an update

useReducer()

- UseReducers takes three arguments:
  - ReducerFn - React calls this function whenever an action triggers.
    - ReducerFn gets latest state snapshot by default and action that is triggered.
    - It returns the new updated state
  - InitialState
  - Initial function

Replacing State

- Lets convert the form state to reducer.
- Add reducer and define the array for storing it.
- Define the reducer function outside of the main function as the data it needs will be passed by react itself.
- Pass the same object as initial object to the reducer.
- But for now this reducerFn will always return the same state, which we do not want.

Replacing State

- Now whenever we require the state to be updated use the dispatcher function which was defined as second element of the array.

  dispatchForm({type:'TITLE',val:events.target.value})
- And in the reducer function add the condition to check the type of action.

  if(action.type==="TITLE"){return {...state,title:action.val}}
- In the similar way dispatch the action for amount and for date as well.

useReducer() with useEffect()

- Check all the instances of formstate and update the code if necessary.
- Now in the useEffect we have used the state as dependency for the callback function.
- Here replace the state with newly created reducer and for that we will add formReducer as its dependency.
- Here comes object-destructuring which is the way to add alias.

  Const {userTitle:titleReducer} = formReducer.

useReducer() vs useState()

- When using state becomes difficult to manage, or you find many bugs and issues during state management you should use reducer.
- State is the main state management tool. Its is used when you have independent piece of data.
- When you have simple state updates you should use state only.
- When you need more power it is great to use reducer.
- When you have related data or dependent data you should consider reducer.

Introduction to context

- When you have many things that are passed using props, or handling many events using props
- You can face issues or it can be complex to handle communication when you forward the data from one component to another.
- Lifting up the state because that component do not have access to the component where the data is needed.
- Here we have another feature called context which is used to trigger actions to pass the data directly to the component where its needed.

Adding context

- We have the concept of context where we do not need such prop chain.
- For that we will add the folder called store or state and will add the file subscriptions-context.js
- We will import React and use React.createContext()

Adding context

- It will have default value of the state you want to maintain component-wide or application-wide.
- `const SubscriptionsContext = React.createContext()`ent.
- Export this const as we will use it in another component.

Providing context

- You need to provide this context to all the components that are going to listen to it.
- Providing means to wrap all the components which are going to tap.
- In JSX code for app component lets wrap components with the <SubscriptionsContext.Provider>
- The second part is listening, we can do that by using context consumer or by using react hook.

Listening context

- Listening to context by using <SubscriptionContext.Consumer> as wrapping element in subscription-list component where we want to listen to the list.
- There we have to define the arrow function between the tags and get the default parameter ctx.
- Copy the whole template in this arrow function.

Listening context

- Where we want to use list we will add ctx.subscriptions.
- But this will not work as we are providing and we only have the default value.
- For that we will add value attribute in provider as value={{subscriptions:[]}}

useContext()

- Instead of using consumer, can use the react hook.
- For that you have to import useContext from react and remove the wrapping element with the arrow function.
- Outside the return statement define const ctx and assign useContext(SubscriptionsContext)
- Now you can use ctx in JSX as we did before

Passing function

- Similar to passing data, can pass the function which will be handler for some user action.
- You just need to add one more property to the context that we maintained
- All the components listening to the context can just use this function to pass data from child to parent.

Passing function

- Can also define the dummy function in the context file as default for IDE
- Generally, when the function is used directly in child component we prefer props and not context.
- Context is only used when forwarding data and creating chain

Custom Provider

- If we do not want to club everything in the app component.
- We can create our own custom provider component
- Add one component in the context file and use props.children
- Can define the states in this context file and also all the handlers.
- As we did in app component pass the value attribute and set the context.
- Now we can wrap the app tag in index file with this newly created component

Context limitations

- Use useContext in the app component if needed. And directly can access the data maintained in the context from all other components.
- There are instances where you need context but not everywhere.
- You should follow the steps for component generation to keep it reusable: Example button.
- Context is not optimized when used for data that changes frequently.
- You should still use props for communication as it is vital and important.

React hooks

- When we say hooks, everything that starts with **use**.
- Hooks should be should in react function component or the custom react hook.
- You cannot use a hook into the nested function for example in useEffect you cannot have useState or useContext.
- For useEffect you should always add the things you use in function as dependency, state update functions are exceptions as they will never change.

Inputs

- Input tag can be made reusable component.
- For that lets add one more component, and name it Input.
- Copy the template from the form component to this and add css
- Now if I want to focus the input in case of invalid value, it would be easier by using ref.

Forward Refs

- So to have such behavior, use ref in the Input component and assign it to the input tag.
- Define the function called focus for inputRef.current.focus()
- Define the ref in form component and pass in the Input tag as **ref**
- But you will get the error as we cannot use ref in our custom components because it is reserved word.

Forward Refs

- useImperativeHandle() to be imported.
- It takes two arguments
  - ref - second argument of the main function
  - Arrow function that returns the object having fields that are accessed- {focus:focus}
- To have this ref accessible you have to use React.forwardRef() as wrapping function to the main function.

Custom hooks

- Already used hooks provided by react.
- Mechanism to create our own hook where we can reuse the logic from different components.
- Extract stateful logic from component.

Why Custom hooks

- Unlike regular function, we can use react hooks in custom hook.
- You can only use react hooks in component functions, so the logic having state updates, or something cannot be used from other component.
- Cannot have common methods which can be used as utils or services.

Creating custom hooks

- Define the folder named hooks.
- Just a .js file - not necessarily first letter capital as in component.
- Define the main arrow function
- The name should start with **use** and shift the common logic as function body.
- Export this function.

Using custom hook

- Showing the time in more than one component at different interval.
- Use custom hook to make this logic reusable.
- As built in hooks returns array, one can return anything from function say state.

Adding parameters

- We want to show time but at different interval.
- For that we have to configure the function.
- Pass the parameter for interval with default value to make it optional
- Parameter as dependency in useEffect

Custom hook with HTTP

- Sending Http request can be made common.
- We call API for get or post or delete.
- Create custom hook having logic to customize the http request.
- Create another .js file and add function to it.

Adding parameters

- Add parameter for passing request configuration
  - Url
  - Method
  - Header
  - Body
- Second parameter to pass function to handle response.

Returning data

- Hook returns loading and error state and returns the function used to call fetch method.
- For that return the whole object.
- You can use the modern javascript syntax for object as we use same attribute and value name.

Calling custom hook

- Using the hook while fetching data.
- Add ternary operators for parameters in custom hook
- For transformation task add arrow function and store it in const.
- Data as default param passed from custom hook.
- Pass the pointer to this function as second argument

Calling custom hook

- Hooks returns the object.
- Apply destructuring in the next line or at same place.
- Add the alias to the function used to call api returned by custom hook.
- By that get rid of whole function but keep useEffect as it is.

UseCallback() with custom hook

- Add the function used to call api as dependency in the useEffect.
- Everytime the app component reevaluates it will call custom hook
- And everytime the function in hook will be created again.
- For that we will wrap the sendRequest method with useCallback

Adjusting parameters

- Add all the values used in the sendRequest function as dependency
- But again every time the component re evaluates the function will be created again.
- For that we need to wrap the transformation function with use Callback in component.
- Remove the config from param and take that in sendRequest function.

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module-5) React - Styling & Advance React

For Styling

- React Bootstrap
- Styled Component

Module-6) React Router

React router

- Library called react router allows us to implement client-side routing
- Install this library using

npm install react-router-dom

Defining routes

- Define routes in app.js
- Can add logic on urls where states are maintained and based on states content gets rendered.
- Import Routes from react-router-dom
- Add two components

Defining routes

```
<Router path= "/home"><Home></Router>
```

- In the root file, import **BrowserRouter** from react-router-dom
- Wrap app component with this in .render()
- Keep all those components in different folder to make it clear.

Navigation

- Add the structure for header and main section in app.js
- Add component header with basic nav and links
- Use this component as header

Link

- Another component to be imported from react-router-dom: **Link**

  <Link to= “/home”>Home</Link>

- Add the styling for different components

NavLink and activeClassName

- The style highlights the link on hover
- But to show active link, one more feature is used: **NavLink**
- Pass the class name using **activeClassName** property

Dynamic routes

- Add the list of users in the user component
- Add another component user-detail
- Define another router for user detail

```
<Router path= “/user-detail/:userId”>
```

- To fetch the dynamic value from url in the component import **useParams** hook.

Switch component

- Update ‘**user-detail/:userId’** to ‘**users/:userId’**
- Use Link component in users component.
- How the details are rendered in the same page?
- Wrap all routes with Switch tag

Exact prop

- Solutions to render single component at a time
  a. Change the order
  b. Add prop to the route tag - **Exact**

Nested Routes

- Routes can be defined in any other component.
- In home component add another route with some other component or any element.
- React will evaluate if any matching path found in component JSX

Redirect component

- No component rendered on domain request.
- To handle this we have another component - **Redirect**
- Add another router with path="/" with exact prop

<Redirect to= "/home">

Page not found

- No matching path found in the routing.
- Add one more component - PageNotFound
- Add this component in app component under path= “*”

Why Authentication?

- Not everyone is having access to the application.
- Not everything is accessible by everyone.

How authentication works?

- In general there are three steps to make authentication work
  - Create an account
  - Login
  - Send request to access resource
- Client sends the request to server for granting permission.

Authentication types

1. Server side session

2. Authentication token

Firebase Authentication

- Firebase will be used as authentication backend.

- Getting started with authentication.
- Select the sign up method

Sign up page

- After adding the new project, add the component for login page
- Toggle the form for login and signup based on state value.
- Add submit Handler
- Call the post request for sign up
- .then to handle the response

Showing warning message

- When response is not **ok** showing up error message.
- Firebase provides message explaining the error.
- One can change the message by parsing the error code.

Login page

- Add the logic for sending the api request to firebase for login
- Even this request is post so using the same code having different url.
- Handle the response for both request in the same manner.

Adding context

- The token that is received from backend needs to be stored common
- For that add another folder - store and auth-context.js
  - Token
  - IsLogged
  - Login(token)=>{}
  - Logout()=>{}

Adding context

- Add the state for token and const for isLoggedIn in provider
- Add handlers for login and logout
- Add const context Value and pass it in the value prop of provider.
- Wrap root component in index.js with this provider.

Using context

- Use context in the login/signup component
- Call login function in the .then block by passing data.idToken
- Adding navigation component with dynamic links based on logged in state

Handling protected Resources

- Api call for changing the password.
- IdToken is expected to be passed as request
- Add the component for profile form having input for new password and submit handler.

Reset password

- Use context to get the idToken to be passed.
- Pass the new entered password and returnSecureToken as false.
- There are different ways of passing IdToken, it depends on your api specifications:
  - As request body
  - As request params
  - Adding authorization header

Improving user experience

- After successful login, redirect user to the profile page.
- Import **useHistory** from react-router-dom
- Call replace function by passing the path of the next page

Logout

- Adding logic to render few pages conditionally.
- For that use context to get the logged in state.
- If user is logged in only then render profile page
- If user is not logged in only then render the login page

Module-6) React Redux

Redux

- What is Redux?
  - A state management system for cross-component or app-wide state

React states

- Types of state
  - Local state - Belongs to single state
  - Cross-component state - Affects multiple component
  - App-wide state - Affects whole app (most of the components)

Redux Vs Context API

- Why redux over context?
- There could be lot of context and context provider
- Complex setup/management
- Performance (not optimized for high-frequency state updates)

How redux works?

- One single store for all the states
- Can be complex to manage such big store, but not needed

Redux core

- Creating store by using .createStore() method of redux.
- Create reducer function.
    - Inputs - Existing state and Dispatched action
    - Output- Updated state
- Pass this function as argument in createStore
- Subscriber function to be passed in .subscribe()

Dispatching action

- On creating store, the counter will increase by one.
- Last step is dispatching the action
- Call .dispatch() function provided by store.
- Can pass the action type as argument

Handling action in reducer

- The goal is to have common reducer for different action
- Different actions perform different kind of operation on data
- Pass action increment
- Test passing decrement action

Installing redux

- Create on component called votes.
- Add the css file
- To use redux install redux using npm install
- Additionally install react-redux

Creating store

- Create folder for store and add .js file
- Import createStore from redux
- Create store and store it in a const.
- Create reducer function and pass it as argument
- We don’t need subscriber here and action dispatch

Providing store

- Provide store to the top of the component tree - **Index.js**
- Import provider from redux
- Wrap the app component with provider.
- Import newly created store and pass it as props

```
<provider store={store}><App/></provider>
```

Subscribing data in component

- Import react hook from react-redux: **useSelector**

  const votes = useSelector(state=>state.votes)

- React redux will implicitly subscribe to the store.

Dispatching actions

- Add two buttons for increasing and decreasing the value.
- For dispatching the action import **useDispatch**
- Define two handlers and dispatch actions from there.
  dispatch({type: 'increment'})

Passing additional data

- One can pass data for extra computation with the type during dispatching of action.
- Increase votes by 10

```
dispatch({type: 'increment',value: '10'})
```

Multiple data

- Show votes only when the button clicked.
- For that add one more property in the state maintained in the store and update all instances
- Add another action **Toggle** to show or hide the votes
- Subscribe to show property using selector and conditionally show the votes

Handling redux state

- The updated state returned from reducer will not merge with existing state
- **Why can’t we just use the state we get as argument?**
- You should never mutate the existing states.
- Also objects are passed as references which can create bugs

Redux Challenges

- Potential Problems:
  - Action types
  - Having many properties
  - State immutability
- Redux Toolkit is another library for handling these problems

Redux toolkit

npm install @reduxjs/toolkit

- Import createSlice and pass the object as argument
  - Name
  - initialState
  - reducers
- Here you can mutate the existing state

Handling redux state

- Define const for created slice(voteSlice)
- Create store by passing voteSlice.reducer
- For multiple slices there will be multiple reducers.
- Import configureStore

```
const store=configureStore({reducer:{votes: voteSlice.reducer}})
```

Defining actions

- voteSlice.actions.increment
- const actions= voteSlice.actions
- To dispatch action, we have to execute the increment() method of actions.
- **Payload** is the name used by toolkit for extra data.

Node JS

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Modules

Module 1 - [Node - NodeJS - Introduction]
Module 2 - [Node – Database –No SQL-Mongodb]
Module 3 - [Node – Introduction to Express JS]
Module 4 - [Structuring REST API]
Module 5 - [Node – Integration with FrontEnd]
Module 6 - [Node – Professional Projects]

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 1

NodeJS - Introduction

Introduction

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!

Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser.

A Node.js app is run in a single process, without creating a new thread for every request.

Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm

When Node.js needs to perform an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

JavaScript (Chrome)
C++
localStorage.getItem
Some C++ function
document.querySelector
Some C++ function
C++ Bindings
JavaScript (Node.js)
C++
fs.readFile
Some C++ function
os.platform
Some C++ function
Chrome
node
Results
JavaScript Code
JavaScript Code
Results

Written in C++

Node JS module system

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

1. Core Modules
2. Own Modules or Local Modules
3. NPM Modules or Third Party Modules

Core Modules

Core Modules covers minimum functionality of Node.js

These core modules are compiled into its binary distribution and load automatically when Node.js process starts

You only need to import the core module first in order to use it in your application

Loading Core Modules

```
var module = require('module_name')
```

- Specify module name in the require() function.

- The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns

Example

Node js File Module System

Own Modules or Local Modules

Local modules are modules created locally in your Node.js application.

These modules include different functionalities of your application in separate files and folders.

You can also package it and distribute it via NPM, so that Node.js community can use it.

We need **module.exports** to define what could be access from other file

Importing our own module Example

NPM Modules or Third Party Module

There are many modules available online which could be used in Node.js.

Node Package Manager (NPM) helps to install those modules, extend them if necessary and publish them to repositories like Github for access to distributed machines

To start working with npm module

1. we have to initialize npm in our application

   npm init

2. we have to install all module we want to use

   npm install validator@version

Validator Example

-Sometimes when download or refer code online there is no **node_modules** directory but it required to run code

npm install

- This command will check json file and download all dependency

- **Also use chalk npm library for practice**

Chalk Example

Global npm module & nodemon

- Search for nodemon in npm it helps to run code automatically as we save code

npm i nodemon@version –g

- Here –g for Global install in OS

nodemon app.js

- To run script

Example

File System & Command line arguments

- We can take input from user using command line arguments

node app.js Ankit

- **process.argv** is Global object like console

console.log(process.argv)

- so we can pass value from command line like what operation we want to perform

node app.js add

Command Line Arguments Example

Arguments Parsing with Yargs

Now in above example

node app.js add –title="This is Title"

This will print same text that we pass through commands

We need to parse the value for that we can use yargs

npm install yargs@version

As shown in code snippet for adding note we can also write code for removing note, listing notes, and reading note similarly

Yargs Example

Storing data with json

- Convert JavaScript object to Json String using JSON.stringify() method.
- Convert Json string to JavaScript object using Json.parse() method

Json Write into File Example

Write Note into File Example

Read data from file

- Read data from file
- **readFile()** does not return actual data

Example

Add | Remove | Save | Load Notes

Example

Add | Remove | Save | Load Notes

Example

Filter vs Find in Javascript

In addNote() code we did previously filter will go through each note from starting to end suppose we have 1000 notes and duplicate note found on 10th position still filter will go through each note till 1000.

So find is better than filter for this situation

```
Const duplicateNote =
        notes.find((note)=>{ note.title===title
})
If(duplicateNote===undefined){
        // add note…
}else{
        // note already exist
}
```

Listing & Reading Notes

Example

Asynchronous Node js

Non blocking event driven

In synchronous programming code executed line by line

Example

Making http request

- register darksky.net/dev

Open Weather Map

npm install request@version

Example

Automatically Json parsing

```
const request = require ('request')
const url='...'
request ( {url:url, json:true }, (error,
        response)=>
        { console.log(response.body.currently)
})
```

Chrome extension json formatter

Example

Geocoding

- Process of taking address and convert it to latitude – longitude
- Mapbox start with free account
- Get token
- API documentation
- Search service -> Geocoding
- Implement in node application similar to temperature API

Example

Error handling

Example

Callback function

EX1

```
setTimeout(()=>{
        console.log('Two seconds are up')
}, 2000)
```

EX2

```
Const names = ['A','B','C']
Const
        shortNames=names.filter((name)=>
        { return name.length<=4
})
```

Example

- callback function required in following example

- instead of return data

```
Const geocode=(address,
        callback)=>{ setTimeout(()=>{
                const data={
                        lat=0, longitude=0
                }
                return data
        }, 2000)
}
Const data=geocode('Philodophia')
Console.log(data)  // error
```

```
Const geocode=(address, callback)=>{
        setTimeout(()=>
                { const
                data={
                        lat=0, longitude=0
                }
                callback(data)
        }, 2000)
}
const data=geocode('Philodophia',
        (data)=>{ Console.log(data)
})
```

Use callback in weather application

Put all code or function inside geocode.js then **module.exports**

In app.js

Const geocode=require(./utils/geocode.js)

Example

Reusable function for getting the forecast

1. setup the forecast function in utils/forecast.js
2. Require the function in app.js and call it
3. the forecast function should have three potential calls to callback
   - low level error, pass string for error
   - coordinate error, pass string for error
   - success, pass forecast string for data

```
forecast(-75.7800, 44.1545, (error,
        data)=>{ console.log('Error',error)
        console.og('Data',data)
})
```

Callback chaining

Example

Get location from user and then geocode

1. Access the command line argument without yargs
2. use the string value as the input for geocode
3. Only geocode if a location was provided
4. Test your work with a couple of locations

Example

ES6 : De-structure object

```
const [latitude, longitude, address] = data
```

Example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 2

No SQL-MongoDB

Mongodb vs Mysql

Mongodb.com

- launch in 2009

- we can use any database with node and we can also use mongo with other technology

SQL Terminology	Mongodb (No-SQL) Terminology
Table	Document
Rows	Collections
Columns	Fields

Installing MongoDB on Windows

Download MongoDB

- We can download any file msi or zip
- Extract downloaded zip folder inside **Mongodb/bin**
- move **Mongodb** folder inside **user/{username}**
- create folder **mongodb_data**
- Open **visual Code** terminal with

mongodb/bin/mongod.exe –dbpath=/users/{username}/mongodb-data
leave terminal running

Install Database GUI viewer

Mongodb Admin tool

Download **Robo 3t not** Studio 3T

Create connection with

Name – Local Mongodb Connection

Address – localhost

Port – 27017

Double click Connection -> Open shell

db.version() => check connection with mongodb database

- It is like Javascript

Connecting to Mongodb

Task-Manager Project
npm i mongodb

Mongodb Driver for Node.js

Ctrl + C to close connection

With mongodb no need to create database

Example

Inserting Document

After Insertion complete

Open robo 3t

Right click **Local MongoDB Connection > Refresh**

- Check data view documents

- **_id** we did not create which is unique id created by mongodb

Example

Insert Bulk (Many)

Ctrl + R to refresh **robo 3T**

Challenge

Insert 3 tasks into new tasks collections

1. Use insertMany to insert 3 documents

- description(String), completed (Boolean)

2. setup the callback to handle error to print ops

3. Run script

4. Refresh the database in robo 3t and view data in tasks collection

Example

The Object ID

- In sql we have id with auto increment behavior
- In mongo _id is generated with GU Id (Global Unique Id)
- Search Mongodb Object Id on Google
- we can use id shown in code to insert collection

```
const {MongoClient, ObjectID} =
require('mongodb')
const id=new ObjectID()
console.log(id)
console.log(id.getTimeS
tamp)
```

- In robo 3t we can see ObjectID(5c….) it is just visualization to easy human readable

```
console.log(id.id.length)     => 12
console.log(id.toHexString().length) => 24
```

- So ObjectID use function internally do process to convert into hexa code

Example

Read Documents from Mongo

If data match with multiple collection then it always return first

Example

find()

find() returns cursor which is pointer

```
db.collection('users').find ( { age : 28 } )
        .toArray( (error, users) => {
                console.log(users)
})
```

- Count data

```
db.collection('users').find ( { age : 28 } )
        .count( (error, count) => {
                console.log(count)
})
```

Challenge

1. Use findOne() to fetch the last task by its id (print document in console)
2. Use find() to fetch all tasks that are not completed (print docs to console)
3. Test your work

Tasks

description : String
completed : boolean

Updating Documents

We can also complete without **updatePromise** variable

Search mongodb update operator for $set

Example

Challenge : updateMany

1. Check the documentation for updateMany
2. setup the call with the query and the updates
3. Use promise methods to setup the success/error handler
4. Test your work

Deleting Documents

Example

Setting up Mongoose

-So far we have done basic operation like create, read, update, delete but how we can decide which field are required compulsory and which are optional and what type of data required

- We can also validate with data type like string, Boolean etc

- Tasks are added by valid user

- We are going add authentication using mongoose

- user1 created some task then user2 not allowed to access those data

- Download from Mongoose.com

Model

- Like real world model which is going to be stored in database
- The methods we use so far in CRUD operation are very low level APIs
- ODM – Object Document Mapping tools in mongoose allows to map objects into Document

```
npm i mongoose
```

- Example

Continue…

- if we set **age:'abc'** then it will throw an error

- **Challenge**

1. Define the model with description and completed fields
2. Create a new instance of the model
3. Save the model to the database
4. Test your work.

Example

Data Validation & Sanitization

- Validation : User age>18
- Alter data before saving – Sanitization
- Example : Remove space around username
- mongoose docs check **Search Validation**

Example

Custom Validator (age > 0)

For Email, Postal code, Credit Card, Pancard we have to use well tested library

npm i validator

Validator Example

Validation Example 2

Schema Types

Check SchemaTypes on Mongoose Docs

Here we can see different validation options like trim, lowercase, default etc.

```
const User=mongoose.model('User', {
        name:{ type : String, required : true, trim : true },
        age : { type : Number, default : 0 },
        email : { type: String, trim : true, lowercase : true }
})
// Test with
const me = new User({ name : '     Ankit     ',
                      email : '   ankit@tops-int.com'})
me.save()….
```

Challenge

Add a password field to User

1. Setup the field as required string
2. Ensure the length is greater than 6
3. Trim the password
4. Ensure that password doesn't contain 'password' (use javascript string includes method)
5. Test your code

Challenge

Task Validation and Sanitization

1. Trim the description and make it required
2. Make completed optional and default it is false
3. Test your work and without error

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 4

Introduction to Express JS

Web server with Express

- Express makes very easy to create web server with node
- It allows to serve all of the assets to web application
- HTML, CSS, Javascript, JSON data

```
npm init –y
npm i express
```

Example

Static Assets

App.js

```
console.log(__dirname)
console.log(__filename)
```

Path module

App.js

```
const path=require('path')
console.log(path.join(__dirname,',,.public/)'))
```

Customize server

Run and Check

App.js

```
const publicDirectoryPath=path.join(__dirname,',,.public/')
app.use(express.static(publicDirectoryPath))
```

Create html pages for about.html and help.html url

Access them directly by typing about.html in url

Serving Css, JS, Images and more

Link CSS and Javascript file with index.html file.

Create public -> img directory

put images inside and display it in index.html

Static Server

Static Assets

Template Engine – Dynamic pages

- Handlebar library
- we will use hbs which is using handlebar in background

  npm install hbs

- web-server -> views -> index.hbs

  app.js

  app.set('view engine', 'hbs')

- **we have inform express about our engine using app.set() method**
- **remove index.html from public directory**

Example

Passing data to hbs file

hbs page also called template

```
app.get(‘’, (req, res)=>{
                res.render(‘index’, {
                        title:’My title’
        })
})
```

index.hbs

```
{{title}}
```

Repeat process for **about.hbs** and **help.hbs** and remove static html pages which are no longer required.

Customizing the views Directory

- What if I want to change views directory name and location how handlebar handle it.
- If change name views directory name then run it show **error**
- If you customize name and location you have to inform express about same.
- Rename (views -> templates)

Example

Handlebars Particles

- Pages we can reuse in multiple other pages like header, footer
- create **header.hbs** inside partical folder with some dummy text

- Example

- restart server automatic by nodemon might not show output as expected
- nodemon only get changes done inside .js file not .hbs file
- use below commad to listen changes of .hbs file

```
nodemon src/app.js –e .js , -hbs
```

Error 404 Pages

- create footer partial page and use inside all views
- set link of all views inside header partial template
- after all request handling inside app.js file
- create 404.hbs view with header & footer

404 – Example

Styles Example

Accessing API from Browser : Query String

to check query string

- console.log(req.query)

```
app.get('/products', (req,
        res)=>
        {
        console.log(req.query.search)
        if( ! req.query.search){
                return res.send({
                                error : 'Your Error message'
                })
        }
```

localhost:3000/products?search=games&rating=5

Weather API Request

Example

Challenge

Goal : Write up /weather

- Copy utils folder from weather-app console project and paste into web-server > src folder

- **npm i request**

1. Require geocode /forecast into app.js
2. Use the address to geocode
3. Use the coordinates to get forecast
4. send back the real forecast request

Default value with De-structuring

```
const transaction = ({type, label, stock=0}={})=>{
        console.log(type, label, stock)
}

transaction('order')
transaction('order','product')
```

Back to weather server app

What happen if we pass **address=!** With query string

- Server will crash

- so that we have to provide default value while de-structuring

Example

Browser HTTP Request with fetch()

- fetch is browser based api not available on node

- Use any dummy api which return JSON data

```
fetch('http://puzzle.mead.io/puzzle')
.then((response)=>{
        response.json().then((data)=>{
                console.log(data)
        })
})
```

Run and check chrome developer console (F12)

Challenge : fetch() weather

1. setup call to fetch to fetch weather for Button
2. Get the parse JSON response
   - if error property, print error
   - if no error property, print location and forecast
3. Refresh the browser and test your work

Example

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 4

Structuring REST AP

Structuring REST API

The Task Resource

Operation	Method	Path
CREATE	POST	/tasks
READ	GET	/tasks /tasks/:id
UPDATE	PATCH	/tasks/:id
DELETE	DELETE	/tasks/:id

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Request - Response

Request

POST /tasks HTTP/1.1
Accept : application
Connection : keep-alive
Authorization : Bearer eyJhGc…
{ description : ‘order new drill bits’ }

Response

HTTP/1.1 201 Created
Date : Sun, 28 Jul 2020
Server : Express
Content-Type : ‘application/json’
{“_id”:”5c13…”, “description”: “Order…”, “completed’ : false}

INSTALL POSTMAN

Resource Creation Endpoints

- npm i nodemon –save –dev
- npm i express
- delete mongodb.js

Example 1

Example 2

Run & Test

npm run dev

package.json

```
"script" : {
        "start" : "node src/index.js",
        "dev" : "nodemon src/index.js"
}
```

Test with post man

Postman>body>raw>json/application

```
{
“name”:”Ankit”,
“email”:”ankit@gmail.com”,
“password”:”Pass123”
}
```

To convert json to object automatically

```
app.use(express.json())
```

HTTP status

- Check httpstatuses.com

- Suppose if we provide password ‘ABC’ then it throws error but status is 200 which is invalid

Index.js

```
.catch((error)=>{
        res.status = 400
        res.send(error)
})
```

Or

```
res.status(400).send(error)
```

Resource Creation Part II

1. Create separate file for task model (load it into index.js)
2. Create the task creation endpoint (handle success and error)
3. Test the endpoint from postman with good and bad data

```
res.status(201).send(user)
```

Represent User Created which is more suitable for API behaviour

Resource Reading Endpoints Part I

Check mongoose documentation for query

Example

Setup end points for Tasks

1. Create an endpoint for fetching all tasks
2. Create an endpoint for fetching a task by its id
3. Setup new requests in postman and test your work

Promise Chaining

Example

Challenge

1. Create promise-chaining-2.js
2. Load in mongoose and task model
3. Remove a given task by id
4. Get and print the total number of incomplete tasks
5. Test your work

Use **findByIdAndDelete()** method

Example

Async / Await

Make more easy Asynchronous promise request

Playground > async-await.js

```
const doWork=()=>{          }
console.log(doWork())
```

If we don’t return anything then by default we get **undefined**

- **async** allows us to create async function in that function we can use **await** feature

```
const doWork = async ()=>{              }
console.log(doWork())
```

- Now it will not return **undefined** instead it will return **Promise** Object

- async function always returns promise, that promise perform the value, we as developer wants to return

```
const doWork = async ()=>{
        return 'Tops Technology'
        }
console.log(doWork())
```

O/P : Promise { 'Ankit' }

```
doWork().then((result)=>
    { console.log(result)
}).catch((e)=>{
    console.log(e)
})
```

- Await operator can be used only inside async function

-When we use async/await we are not changing internal structure of Promise instead we are changing the way we work with it

- Mongoose support promises so as a consumer we can use async/awat to replace promises

-Example

- async / await doesn’t make things faster it makes things easier

- Another problem with Promise chaining is difficult manage all value in single scope

- means we can not access sum , sum1, sum2 all at one place

- what if one of Promise reject in async/await

```
Const add = (a, b)=>{
        return new Promise((resolve,
                reject)=>{ setTimeout(()=>{
                        if(a<0 || b<0){ return reject(‘Numbers must be positive’)}
                        resolve(a+b)
                }, 2000)
        })    }
```

Use Async/Await with Task-Manager

Example Patch & Delete & Post & etc

Challenge

1. Create deleteTaskAndCount as an async function
   - Accept id of task to remove
2. Use await to delete task and count up incomplete tasks
3. Return the count
4. Call the function and attach then/catch to log results
5. Test your work

Integrating Async/Await

Index.js

- **Similarly change each use API with Async/await**
- **and also for each task api**

```
App.post('/users', async (req,
    res)=>{ const user=new
    User(req.body) try{
        await user.save()
        res.status(201).send(user)
    }catch(e){
        res.status(400).send(e)
    }
})
```

Challenge

Update Task By Id

1. Setup the route handler
2. Send error if unknown updates
3. Attempt to update the task
    - Handle task not found
    - Handle validation errors
    - Handle Success
4. Test your work

Separate route files

- Move all user routers to routers/user.js and also fix all required methods

- rename app with router

Challenge

1. Create new file that creates/exports new routers
2. Move all the task routes over there
3. Load in and use that router with the express app
4. Test your work

Example

API Authentication & Security

Securely Storing Password :

npm bcrypt.js

- Hashing is one way algorithm

- Encryption algorithm we can also turn password back to original algorithm

Example

Comparing password when login

```
const isMatch = await bcrypt.compare('Red12345', hashedPassword)
console.log(isMatch)
```

Mongoose Middleware

- Before save user encrypt password
- Middleware specified on the schema level
- we are going to use document Middleware

- Example

Update operation code change

Challenge

1. Find the task

2. Alter the task parameter

3. Save the task

4.Test your work by updating a task from postman

Routers/user.js

```
router.patch('/users/:id', async (req, res)=>{
        …
        try{ const user= await User.findById(req.params.id)
        updates.forEach((update)=> {
                user[update]=req.body[update]
        })
        await user.save()
        }catch(error){ … }
})
```

Logging users

- Before test above code we have to confirm with email must be unique for that set **unique:true** with userSchema

- And drop database to make work it on

- Now test it in postman try to save 2 users with same email

- Then test login url

Example

JSON Web Tokens

- Authenticated User can only delete task or other task related operations

- After login user get Token and that token will be helpful for that user to perform other task like create new entry, delete, …

npm jsonwebtoken

- Json token made of 3 parts

Seprated by dot(.)

1st part – base64 encoded string

2nd part – base64 unique Id we provide

Copy it and check on base64decode.org

Example

Verify Token and set expiry

Verify Token ->

```
const data = jwt.verify (token, ‘thisismytoken’)
console.log(data)
```

We can also set expiry

Test with {0 seconds}

```
const token = jwt.sign({ _id : ‘abc123’ }, ‘thisismytoken’,
        {expiresIn : ‘7 days’ })
```

Generating Authentication Tokens

- here user.generateAuthToken() is instance method not static method like findByCredential()

- Test the code with json

- Now we have generated token but it has not been saved so we can not compare for next request, so we have to store the token once it is generated

Example

Store token

Test code with login url

The same operation we need to perform when we create new user

Challenge

1. When create new user generate token and return with response
2. Also store token in database

Test the code with create user

Models/user.js

```
Const userSchema = mongoose.Schema({
        name : { … }
        email : { … }
        tokens : [{
                token : { type:String, required : true }
                }]
})
userSchema.methods.generatedToken = async
        function(){ const user=this
        const token = …
        user.tokens=user.tokens.concat({token})
        await user.save()
        return user
}
```

Express Middleware

- Now we have to use token return after login or create user to authenticate user for other request

Every request require auth except signin and create user (signup)

Without Middleware

new request -> run route handler

With Middleware

new request -> do something -> run router handler

Test code it will not Complete the

Request without next()

Example

Index.js

```
app.use((req, res, next)=>{
        console.log(req.method, req.path)
        next()      // required to complete request
})
```

Accepting Authentication Token

Test the code Get Users

Pass headers with

Authoriization=“Bearer 5cToken“

- we can also validate user to read only his data instead of all users information

req.user=user

next()

Logging out

Test the code

-To logout from all devices

Remove all tokens and save user

- Setup POST users/logoutAll

req.user.tokens=[]

await req.user.save()

Hiding Private Data

Instead of returning use return only public profile

res.send({ user: user.getPublicProfile() })

- we can also use **toJson** instead of **getPublicProfile** name of the function

- It will behave same as **toString**() in Java

-when we return an object or print object this method invokes automatically and we can return whatever we want to.

```
userSchema.methods.getPublicProfile=function(){
        const user = this
        const userObject=user.toObject()
        delete userObject.password
        delete userObject.tokens
        return userObject
}
```

```
userSchema.methods.toJson=function(){
        .....
}
```

Email Sending :

Install nodemailer module : npm i nodemailer

- Check out EmailEngine – a self-hosted email gateway that allows making REST requests against IMAP and SMTP servers. EmailEngine also sends webhooks whenever something changes on the registered accounts.
- Using the email accounts registered with EmailEngine, you can receive and send emails. EmailEngine supports OAuth2, delayed sends, opens and clicks tracking, bounce detection, etc. All on top of regular email accounts without an external MTA service.

- Example :

Chat app :

Install socket io module : npm i socket.io

- Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server.
- It is built on top of the WebSocket protocol and provides additional guarantees like fallback to HTTP long-polling or automatic reconnection.
- Example :

File Uploading:

Install multer module : npm i multer

- Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.
- NOTE: Multer will not process any form which is not multipart (multipart/form-data).
- Example :

Payment:

payment with razorpay : npm i razorpay

- Example :

TOPSTECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Module 5

Integration with Front End

1. Set Up the Node.js Server
2. Develop Your Frontend
3. Handling CORS (Cross-Origin Resource Sharing)
4. Testing and Debugging
5. Deployment

CI /CD Module

- ✓ Continuous Integration (CI) is a software development practice that encourages the frequent merging of code changes from multiple contributors into a single project.
- ✓ *This process helps to reduce errors and increase the speed of development.*
- ✓ *Basic Continuous Integration and Continuous Deployment (CI/CD) pipeline for deploying a Node.js application in a DigitalOcean environment.*
- ✓ *CI/CD is a fundamental practice for automating the software development and delivery process, enabling teams to deliver their applications faster and more reliably*

CI /CD Module

Step 1: Configuring the Git Repository

- ✓ Create a new Git repository for your Node.js application on GitHub or any other Git hosting service.
- ✓ Initialize a new Node.js project on your local machine using the npm init command.
- ✓ Implement your Node.js application.
- ✓ Initialize Git in the root directory of your project using the git init command.
- ✓ Add all files to Git using the git add . command and make the initial commit using git commit -m "Initial commit".

CI /CD Module

Step 2: Setting Up the CI/CD Pipeline with GitHub Actions

- In your GitHub repository, go to the “Actions” section and click on “Set up a workflow yourself” to create a new configuration file.
- Copy the following basic content into the .github/workflows/ci-cd.yml file:

CI /CD Module

Step 3: Testing the CI/CD Pipeline

- Make a change in your code locally.
- Commit and push the changes to the Git repository.
- Access GitHub Actions and verify if the CI/CD pipeline is automatically triggered.
- After the pipeline completes, check if your application is successfully deployed.