**Full-Stack Web Application:**

**Day01:** **Full-Stack Web Application:**

**UI Vs UX:**

=========

UI ==> User Interface

the users can have a platform to make interactions with application

images, hyperlinks, tables, list etc.

also called as "front-end".

UX ==> User Experience

Banking application

transactions

clicks ==> working properly or not.

**FULLSTACK APPLICATION ARCHITECTURE:**

**==================================**

Including with three layers:

**1) Front-end layer/UI Layer:**

**============================**

client side

user a device (smart phone, laptop, smart tv) + browser

user ===> application (server)

<==================

login ==>

mobile/mail

pwd

==> client is for sending the request to the server and receiving the response from the server.

**Browser:**

**=======**

application

required to send and receive request and response to/from server.

Chrome ==> Google

Safari ==> Apple

Internet Explorer/Microsoft Edge etc.

**Backend:**

**=======**

2) WebLogic layer

3) Database layer ==> storing data

User ===> Browser ===> Request ==> Web logic layer

Ex: login data

user ==> request(mail/mobile and pwd (chrome)) <==> server (web logic layer <====> Database layer)

**Server:**

**======**

==> an application

it can receive the user request

provide the user response

manipulations

**For the full stack development:**

**===============================**

front end and back end

full stack developers ==> must have the knowledge of end to end application development

**front-end**

**=========**

HTML ==> Hyper Text Markup Language

CSS ==> Cascading Style Sheet

Bootstrap

JavaScript

JavaScript library: React/Angular/Node etc.

**Backend:**

**=======**

web logic layer ==> PHP/C++/Python/Java/.net/node Js etc.

Database layer ==> SQL

Database

two types:

1) SQL based db: MySQL Server, Oracle

2) No SQL based db: Mongo DB, Postgre etc.

**Day02:** **Fundamentals of UI Technologies:**

**HTTP:**

**====**

==> Hyper Text Transfer Protocol

==> Protocol ==> set or rules

==> HTTP protocol describe about the:

format of Request

format of Response

client ===========> request ===========> server

<=============================================

Response

==========================================

**WEB APPLICATION:**

**================**

**APPLICATION:**

**============**

==> Collection of programs for multiple events

ex: WhatsApp

{

check the contacts

can post the status

can read/check other's status

texting

audio calling

video calling

banking transactions

}

==> Application (app) can always run on platform (OS)

ex: WhatsApp, Phonepe, Instagram etc. ==> Mobile applications

mobile dependent platforms/os

Android, IoS

ex: notepad, word, excel etc. ==> OS (Windows)

**Software:**

**========**

==> Collection of programs for multiple events

Ex: Windows, Linux, Android etc.

==> Applications are classified into three types:

**1) Standard Applications:**

**=========================**

we need a platform like os

can use in only one device at a time.

Ex: Notepad, Calculator etc.

**2) Web Applications:**

**====================**

The applications can access through internet called as "Web applications"

ex: Gmail, Youtube, Amazon etc.

**3) Distributed Applications:**

**============================**

Ex: Phonepe ==> 3rd party application

Phonepe ==> Check my account Balance

Phonepe =============> Our bank application/server (ICICI)

User: UPI Id

UPI PWD

<=================

Response:

My account Balance

==> An application can interact with other application to accomplish any task is called "Distributed Application".

**For UI Development:**

Three major technologies:

1) HTML ==> Hyper Text Transfer Protocol

we can able display any web content

2) CSS ==> Cascading Style Sheets

we can define more style to web content

3) JavaScript.

**Day03: Introduction To HTML:**

Website ==> Collection of web pages

Web page ==> Collection of Web elements

Web Element ==> an HTML Element

**HTML Introduction:**

**==================**

**HTML** ==> Hyper Text Markup Language

==> Syntax of HTML is completely with "tags".

Syntax for tag:

<element-name/text>

ex: <a> ==> a tag ==>

<p> ==> p tag

<ravi> ==> ravi tag

==> HTML we can use to develop web pages.

Webpages ==> web elements like: Links, paragraphs, headings, images, list, tables etc.

==> For the webpage/website: the appearance can be defined with HTML.

Java/Python ==> Syntax based programming

Syntax ==> Statement

**Limitation:**

**============**

we can't define styles to the webpage/website using an HTML.

Styles:

padding, border, margins, background, color etc.

**Markup Language:**

**===============**

the language with tags

ex: HTML, WML, SGML etc.

**Hyper Text:**

**===========**

we can make the transfer from one page of text to another page of text

or

one server to another server.

**HTML Features:**

**==============**

1) Easy language

2) HTML code execute on the client (browser)

3) Can support all the browsers.

4) Case Insensitive Programming language.

ex: <a> ==> lower case text

<A> ==> upper case text

5) Standard Programming language

W3C ==> World Wide Web Consortium

6) HTML file can created with an extension of ".html"

Syntax:

file-name.html

7) Interpreter dependent language.

**Translators:**

**============**

High level instructions ==> low level instructions

Compiler

Interpreter

program:

{

====

println();

array();

====

}

==> compiler, will divide the total amount of code into multiple blocks

definition{

1

2

3

}

println(){

1

2

3

4

}

array()

{

1

2

3

4

}

interpreter ==> cannot divide the total program into multiple blocks

=============================================================

**CSS:**

**====**

==> Cascading Style Sheets

==> Property based language.

that every property in CSS must be with at least one value

**Syntax:**

property-name : value;

Ex: color : blue;

padding : 2px;

==> CSS can use to make the HTML document (web document) more beautiful.

**Limitation:**

**===========**

The CSS and HTML cannot able to define the functionality.

**JavaScript:**

**===========**

to the web app, define the functionality

"JavaScript" can be used.

=======================================

**History of HTML:**

**================**

==> Tim Berners Lee

1991

Different Versions:

HTML 1.0 ==> 1991

HTML 2.0 ==> 1995

HTML 3.0 ==> 1997

HTML 4.0 ==> 1997

HTML 5.0 ==> 2014

HTML 5.1 ==> 2016

HTML 5.2 ==> 2017

HTML 5.3 ==> 2020

=============================================

**HTML Program Architecture:**

**==========================**

<!DOCTYPE html>

<html> ==> start point

<head>

External Information which is for relevant to the browser

</head>

<body>

main content, what we want to display as web content

</body>

</html> ==> close point

**HTML tags:**

classified into two types:

**1) Paired Tags:**

must be with open and close tags

Syntax:

<open> Context </close>

ex: <p> Kalyan Gudala </p>

**2) Unpaired Tags:**

should be with only open tags

==> self-closing tags.

Syntax:

<open tag>

Syntax:

<!Doctype>

<img> etc.

==> doesn't require any content to display.

**<!Doctype> tag:**

**==============**

==> Unpaired tag

used to specify the type of the markup language.

Syntax:

<!Doctype html>

==> **To write any HTML code:**

three steps:

1) Create the HTML file

2) Write the HTML Code

<!DOCTYPE html>

<html>

<head>

<title> My First Web Development </title>

</head>

<body>

Hi

Good Morning

Welcome To Ashok IT.

</body>

</html> ==> close point

3) Run the HTML code

**Day04: Basic HTML Tags Part\_01:**

**IDE:**

**====**

==> Integrated Development Environment

**VS Code:**

**========**

Visual Studio Code

**VS Code Download and Installation:**

**===================================**

Browser ==> Search for: VS Code ==> Click on the Download VS Code Link ==> Based on your OS, you can start download

==> After the downloading:

by simply click on that downloaded file, we can start the installation.

Accept the agreement (radio button)

next

next

next

Select: Desktop Short cut

next

Install

**=========================================**

**Creation of Project Folder:**

**===========================**

We should attach this project folder to the VS Code

<!DOCTYPE html>

<html>

<head>

<title> Greetings </title>

</head>

<body>

Good Morning.

</body>

</html>

===================================================

**Basic HTML Tags:**

**================**

**1) Headings:**

**============**

==> total 6-levels of headings

for headings and sub-headings

<h1> </h1>

<h2> <h2>

<h3> </h3>

<h4> </h4>

<h5> </h5>

<h6> </h6>

**Note:**

**=====**

==> all heading tags are block elements

<!DOCTYPE html>

<html>

<body>

<h1> UI Development </h1>

<h2> HTML </h2>

<h3> Headings In HTML </h3>

<h4> H1 to H6 Headings </h4>

<h3> Paragraphs In HTML </h3>

<h4> P Tag for Paragraph </h4>

<h3> Links </h3>

<h4> a tag for Hyper links </h4>

<h2> CSS </h2>

<h3> Selectors </h3>

<h3> Properties </h3>

<h2> JavaScript </h2>

<h3> Bootstrap </h3>

<h3> React JS </h3>

<h3> Angular </h3>

<h3> Node </h3>

</body>

</html>

==========================================

**Attributes:**

**===========**

==> Are extra information for HTML Tags.

Ex:

<img src = "ravi.jpg">

**Note:**

**=====**

No attributes for Heading tags.

==========================================

**Paragraph:**

**==========**

**<p> tag:**

**========**

Syntax:

<p> Paragraph </p>

**Note:**

**====**

Block element

No Attributes are for <p> tag.

**HTML Elements:**

**==============**

<tag> Content </tag>

==> can be classified into two types:

1) Inline HTML Elements ==> the result of inline elements always with same line.

2) Block HTML Elements ==> any block element can define the content in new line only for every definition.

Ex:

<t> Kalyan </t>

<t> Srikar IT </t>

Kalyan Srikar IT

=============================

<s> Kalyan </s>

<s> Srikar IT </s>

Kalyan

Srikar IT

<!DOCTYPE html>

<html>

<head>

<title> Paragraphs in HTML </title>

</head>

<body>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

</p>

<p>

It is a long-established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humor and the like).

</p>

</body>

</html>

**Assignment:**

**==========**

1) https://code.visualstudio.com/docs/?dv=win64user ===> for only headings

2) https://www.lipsum.com/ ==> headings and paragraphs

**Day05: Basic HTML Tags Part\_02:**

**<br> Tag:**

**=========**

==> Line Break Tag

==> Self closing tag

we can use this to get the new line

**Syntax:**

<br> or <br/>

==> Void HTML Element.

If any tag cannot accept any content/context to make display, those are called as "Void Elements"

<p> context </p> ==> Non-void Elements

==> <br> tag also does not have any attributes.

<!DOCTYPE html>

<html>

<head>

<title> Dummy Context </title>

</head>

<body>

<p>

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.<br><br>

The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, accompanied by English versions from the 1914 translation by H. Rackham.

</p>

</body>

</html>

<html>

<p>

context-1<br><br>

context-2 <br><br>

</p>

</html>

=====================================

**<div> tag:**

**==========**

==> division tag

==> blocked container

**Syntax:**

<div> Content </div>

<div> A </div>

<div> B </div>

<div> C </div>

A

B

C

<!DOCTYPE html>

<html>

<head>

<title> Division Tag </title>

</head>

<body>

<div> Kalyan </div>

<div> Srikara IT </div>

<div> Fullstack Developer </div>

<div>

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration in some form, by injected humour, or randomized words which don't look even slightly believable. If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything embarrassing hidden in the middle of text. All the Lorem Ipsum generators on the Internet tend to repeat predefined chunks as necessary, making this the first true generator on the Internet. It uses a dictionary of over 200 Latin words, combined with a handful of model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-characteristic words etc.

<br><br>

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

</div>

</body>

</html>

========================

==> Real meaning of the div tag is:

to develop the webpage with multiple sections

==> sectional development within the webpage:

<div>

headings

paragraphs

images

links

tables

</div>

<!DOCTYPE html>

<html>

<head>

<title> Srikara IT Replica </title>

</head>

<body>

<div>

<h2> Elevate Your Career with Industry-Ready IT Courses </h2>

<div>

<div>

<h1> JAVA </h1>

<p> Full Stack Development </p>

</div>

<div>

<p> Course Duration : 5 months <br><br>

Offer Price: 30000/-

</p>

</div>

</div>

</div>

</body>

</html>

======================================================

**<hr> Tag:**

**=========**

==> Horizontal Rule Tag

**section-1**

**================**

section-2

==> To get the horizontal line between two sections/two elements, we can use <hr> tag.

==> Self-closing tag

==> Void Element.

Syntax:

<hr>

<!DOCTYPE html>

<html>

<head>

<title>Srikara IT Replica </title>

</head>

<body>

<div>

<h2> Elevate Your Career with Industry-Ready IT Courses </h2><hr>

<div>

<div>

<h1> JAVA </h1>

<p> Full Stack Development </p>

</div>

<div>

<p> Course Duration : 5 months <br><br>

Offer Price: 30000/-

</p>

</div>

</div>

<hr>

<div>

<div>

<h1> PYTHON </h1>

<p> Full Stack Development </p>

</div>

<div>

<p> Course Duration : 4 months <br><br>

Offer Price: 18000/-

</p>

</div><hr>

</div>

</div>

</body>

</html>

<h1> Web Development <h2> HTML </h2> </h1> ==> In correct

<p> Paragraph <h2> chchch </h2> </p> ==> Incorrect

<p> context </p> context </p> </p> ==> Incorrect

**Day-06: Text Formatting Tags:**

**Text Formatting Tags:**

**=====================**

==> used to format the text

formatting the text : bold format, italic format, underline format etc.

**Note:**

**=====**

all text formatting tags are inline elements.

**1) <b> tag:**

**==========**

==> b == bold

==> paired tag

which we can use to display the text as bold.

Syntax:

<b> Context/text </b>

**2) <strong> tag:**

**================**

==> <b> and <strong> tags ==> working as same

==> <strong> can use to define the boldness to the text schematically.

==> screen readers/blind people, <strong> can be used.

==> paired tag

Syntax:

<strong> Context </strong>

**3) <i> tag and <em> tag:**

**========================**

i => italic tag

em ==> emphasis tag

==> <i> and <em> ==> with same functionality

<i> ==> for important text

<em> ==> screen readers

==> Paired Tags

Syntax:

<i> Context </i>

<em> context </em>

**4) <u> tag & <ins> tag:**

**=======================**

<u> ==> Underline

<ins> ==> Inserted

Ex: HTML is abbreviated as ----------

==> Paired Tags

Syntax:

<u> Context </u>

<ins> Context </ins>

**5) <strike> tag and <del> tag:**

**==============================**

<strike> ==> used to strike off text

normal text representation

<del> ==> used to strike off/delete the text

schematically.

when the text/data got deleted permanently from the server

to represent these kind of text:

<del> tag.

Syntax:

<strike> Context </strike>

<del> context </del>

**6) <mark> tag:**

**==============**

==> paired tag

can be used to highlight the text with yellow background

Syntax:

<mark> context </mark>

**7) <sup> tag:**

**=============**

<sup> ==> superscript

can be used to display the text in superscript format (slightly above the text line>

Syntax:

<sup> Context </sup>

**8) <sub> tag:**

**=============**

subscript tag

can use to display the text as subscript (slightly below to the text line)

Syntax:

<sub> context </sub>

<!DOCTYPE html>

<html>

<head>

<title> Text Formatting </title>

</head>

<body>

<h1> <u> Text Formatting in HTML </u> </h1>

<b> UI Development </b>

<b> HTML </b>

<b> CSS </b>

<b> JavaScript </b>

<div>

<p>

Fullstack <b> UI </b> Development<br>

HTML<br>

<b> CSS </b> <br>

JavaScript <br>

<b> React JS </b> <br>

Node JS <br>

Express

</p>

</div>

<strong> Java Fullstack </strong>

<strong> Python Fullstack </strong>

<p>

<b><i> Ashok IT </i></b> from <i> Hyderabad </i>, we offer:<br>

Java <em> Fullstack </em><br>

<em> Python </em> Fullstack <br>

<i> Generative AI </i><br>

<em> AWS Devops </em> <br>

UI Fullstack <br>

Automation Testing <br>

Data Science

</p>

<p>

HTML is abbreviated as <ins> Hyper Text Markup Language </ins><br>

CSS is abbreviated as <ins> Cascading Style Sheets </ins>

</p>

<strike> .net Development </strike><br>

<del> GCP Devops </del>

<mark> Srikara IT </mark><br>

<p>

X<sup>3</sup>-3X<sup>2</sup>+7X-9;

</p>

<p>

H<sub>2</sub>O<br>

H<sub>2</sub>SO<sub>4</sub>

</p>

</body>

</html>

===============================================================

**DAY07: Images in HTML:**

**IMAGES IN HTML:**

**===============**

**path/URL:**

**=========**

URL ==> Uniform Resource Locator

Ex: C:\Users\Sai Srikar\Python-UI-Development\UI\_Development\_HTML\_CSS\Day\_03

==> URLs can be represented in two ways:

**1) Absolute URL/Full URL:**

when the target file is outside of the project folder (from other folder or from other website...), we can use "absolute path".

Ex: C:\Personal\Ashok IT\UI Batches\UI\_9am\UI Development\_HTML\_CSS\Day\_03\Text Formatts

**2) Relative URL/Short Path:**

when the target file is within the project folder, we can use the "relative url".

ex: file-name.extension

Ex:

image

P1 P2

==> To make display an image as web content:

<img> tag can be used.

**<img> tag:**

**==========**

1) Unpaired Tag

doesn't need the close tag

2) Self-closing tag

3) Inline Element

4) Void Element.

**Syntax:**

<img src = "Absolute URL/Relative URL of the image location">

Pixels.com ==> Official website

from where we can collect the images

**Image with Relative Path:**

**=========================**

<!DOCTYPE html>

<html>

<head>

<title> Images with Absolute Path </title>

</head>

<body>

<img src = "sampleImage.jpg">

</body>

</html>

**Image with absolute Path:**

**=========================**

<!DOCTYPE html>

<html>

<head>

<title> Images with Absolute Path </title>

</head>

<body>

<img src = "C:\Users\DELL\Downloads\SampleImage.jpg">

</body>

</html>

**Attributes:**

**===========**

**1) width:**

**========**

==> to adjust the image horizontally (x-direction), width attribute can be used.

Syntax:

<img src = "path" width = "value">

to represent the width:

units: px (pixels), %, em

**2) height:**

**===========**

==> to adjust the image vertically (y-direction), height attribute can be used.

Syntax:

<img src = "path" height = "value px|%|em">

<!DOCTYPE html>

<html>

<head>

<title> Images with Absolute Path </title>

</head>

<body>

<img src = "C:\Users\DELL\Downloads\SampleImage.jpg" width = "400px" height = "400px">

</body>

</html>

**3) style:**

**=========**

==> we can define both height and width to the image.

Syntax:

<img src = "path" style = "height : value;width : value">

<!DOCTYPE html>

<html>

<head>

<title> Images with Absolute Path </title>

</head>

<body>

<img src = "sampleImage.jpg" style="height: 450px;width: 300px;">

</body>

</html>

**4) alt:**

**========**

==> alternative text

==> when the image is not found,

to make the display some alternative text

"alt" attribute.

Syntax:

<img src = "path" alt = "alternative text">

<!DOCTYPE html>

<html>

<head>

<title> Images with Absolute Path </title>

</head>

<body>

<img src = "C:\Users\DELL\Downloads\SampleImag.jpg" width = "400px" height = "400px" alt="Image Not Found in the given repository.">

</body>

</html>

==========================

**5) title attribute:**

**===================**

==> can work while hovering (placing the cursor on the element)

==> while the hovering: to get the title of the image, we can use "title" attribute.

Syntax:

<img src = "path" title = "title">

<!DOCTYPE html>

<html>

<head>

<title> Images with Absolute Path </title>

</head>

<body>

<img src = "sampleImage.jpg" style="height: 450px;width: 300px;" title="Ravi">

</body>

</html>

=========================================

**Display the image from other website:**

==========================================

<!DOCTYPE html>

<html>

<head>

<title> Home </title>

</head>

<body>

<h1> Ashok IT </h1>

<div>

<img src = "https://www.ashokitech.com/assets/banners/Software%20Testing.jpg" style = "height: 20%; width: 100%;">

</div>

<div>

<h2> Elevate Your Career with Industry-Ready IT Courses </h2>

<div>

<img src = "https://www.ashokitech.com/assets/uploads/course/Java%20full%20stack%20development.jpg" style="height: 300px;width: 30%;">

<img src = "https://www.ashokitech.com/assets/uploads/course/Java%20full%20stack%20development.jpg" style="height: 300px;width: 30%;">

<br>

<p>

course Duration : 6 months <br>

Offer Price : 30000/- <br>

<button> View More </button>

</p>

</div>

</div>

</body>

</html>

**DAY08: Hyper Links:**

**Hyperlinks:**

**===========**

==> are used to navigate from one website to another website or

one web page to another web page or

one section to another section.

==> <a> tag

anchor tag

**<a> tag:**

**=========**

==> paired tag

Syntax:

<a> Context </a>

==> non-void element

==> Inline element.

**Note:**

**=====**

how the <p> or <div>

text in <p>

text in <div>

like that, <a> tag also.

==> To make the normal text of <a> tag as link text:

attribute:

href ==> Hyper Reference

<!DOCTYPE html>

<html>

<head>

<title> Anchor Tag </title>

</head>

<body>

<h1> Anchor Tag Behavior </h1>

<a href> Text\_1 </a><br>

<a href> Text\_2 </a><br>

<a href> Text\_3 </a><br>

</body>

</html>

**href attribute:**

**===============**

href ==> navigation place

href can define with:

absolute path

relative path

**Links with absolute path:**

**=========================**

Link ==> Blue (before the performing of click action)

Link ==> Purple (After the performing of click action)

for the links:

**states:**

**1) Link state:**

**===============**

==> the link is not performed with any click action

are called as "links"

==> active links ==> blue (by default)

**2) Visited state:**

**==================**

==> the link which was already used

are called as "visited links".

==> visited links ==> purple (by default)

**3) Active state:**

**=================**

==> the link in click (during the click action)

==> active state ==> red (by default)

**4) Hover State:**

**==============**

==> placing the cursor on link

**Links with Relative Paths:**

**==========================**

<!DOCTYPE html>

<html>

<head>

<title> Home </title>

</head>

<body>

<div>

<div>

<a href = "home.html"> Home </a>

</div>

<div>

<a href = "services.html"> Services </a>

</div>

<div>

<a href = "contact.html"> contact us </a>

</div>

<h1> Ashok IT </h1>

<div>

<img src = "https://www.ashokitech.com/assets/banners/Software%20Testing.jpg" style = "height: 20%; width: 100%;">

</div>

<div>

<h2> Elevate Your Career with Industry-Ready IT Courses </h2>

<div>

<img src = "https://www.ashokitech.com/assets/uploads/course/Java%20full%20stack%20development.jpg" style="height: 300px;width: 30%;">

<img src = "https://www.ashokitech.com/assets/uploads/course/Java%20full%20stack%20development.jpg" style="height: 300px;width: 30%;">

<br>

<p>

course Duration : 6 months <br>

Offer Price : 30000/- <br>

<button> View More </button>

</p>

</div>

</div>

</body>

</html>

=============================================

**Attributes in <a>:**

**==================**

1) href

2) title = "any title"

3) target = "\_self/\_blank"

**DAY09: Images as Links, Bookmarks:**

**Images as links:**

**================**

Syntax:

<a href = "navigation\_link" target = "\_self/\_blank"> <img src = "Image Path\_Absolute\_Relative" style = "height:value;width:value;"> </a>

<!DOCTYPE html>

<html>

<head>

<title> Images as Links </title>

</head>

<body>

<h1> ECommerce Application </h1>

<a href = "https://www.amazon.in/s?bbn=81107433031&rh=n%3A81107433031%2Cp\_85%3A10440599031&\_encoding=UTF8&content-id=amzn1.sym.58c90a12-100b-4a2f-8e15-7c06f1abe2be&pd\_rd\_r=5d24cfab-75b3-4d62-bc76-b13ffa933be6&pd\_rd\_w=UmrzP&pd\_rd\_wg=lCscJ&pf\_rd\_p=58c90a12-100b-4a2f-8e15-7c06f1abe2be&pf\_rd\_r=81H11DKV8G8N18154WZ6&ref=pd\_hp\_d\_atf\_unk" target="\_blank"> <img src = "image\_pract.jpg" height="400px" width="400px%"> </a>

</body>

</html>

====================================================================

Bookmarks:

==========

==> when we need to navigate from one section of ebook document to another section, we can use "bookmarks".

==> bookmarks are always be defined with "<a> tag" only.

==> href for <a> should be "id" value.

Syntax:

<a href = "#id\_value"> Link Text </a>

**id:**

**===**

==> an attribute

==> we can use to uniquely identify different sections of webpage/web application.

==> id should always define with unique values.

<!DOCTYPE html>

<html>

<head>

<title> Bookmarks in HTML </title>

</head>

<body>

<div id = "Unit-1">

<h1> UNIT-1: Introduction </h1>

<div>

<h2> What is Lorem Ipsum? </h2>

<p>

<a href = "#Unit-3"> Unit-3 </a><br><br>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularized in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

</div>

<div>

<h2> What is Lorem Ipsum? </h2>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

</div>

</div>

<div id = "Unit-2">

<h1> UNIT-2: Introduction </h1>

<div>

<h2> What is Lorem Ipsum? </h2>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

</div>

<div>

<h2> What is Lorem Ipsum? </h2>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

</div>

</div>

<div id = "Unit-3">

<h1> UNIT-3: Introduction </h1>

<div>

<h2> What is Lorem Ipsum? </h2>

<p>

For more understanding please refer here:<br>

<a href = "#Unit-1"> Unit-1 </a><br><br>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

</div>

<div>

<h2> What is Lorem Ipsum? </h2>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

<p>

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum

</p>

</div>

</div>

</body>

</html>

Note:

=====

when we don't have the navigation path while defining the <a> tag:

href = "#"

**Day10:**  **List Part\_01:**

LISTS:

======

==> REQUIRED TO REPRESENT THE DATA IN POINT NOTATION.

==> CAN BE REPRESENTED IN THREE WAYS:

1) ORDERED LIST

2) UNORDERED LIST

3) DESCRIPTION/DEFINITION LIST

ORDERED LIST:

=============

==> to make display the content in the specific order.

==> tag: <ol>

<ol> tag:

paired tag

block element

Syntax:

<ol>

</ol>

==> to specify the content in <ol> tag:

tag: <li> tag

==> Paired Tag

<li> List Item </li>

==> Block Element

==> to develop the ordered list:

we can nest the <li> within the <ol>

Syntax:

<ol>

<li> list item-1 </li>

<li> list item-2 </li>

</ol>

Note:

====

==> the <li> in <ol> can display with decimal number by default.

==> we can customize this using the CSS.

<!DOCTYPE html>

<html>

<head>

<title> Ordered List </title>

</head>

<body>

<h1> UI Development </h1>

<h2> Content Order </h2>

<ol>

<li> HTML (Hyper Text Markup Language) </li>

<li> CSS (Cascading Style Sheet) </li>

<li> JavaScript </li>

<li> Bootstrap </li>

<li> React JS </li>

</ol>

</body>

</html>

=============================================

Unordered List:

===============

==> the data without any specific order can be represented with "<ul> tag.

<ul> tag

========

paired tag

Syntax:

<ul> </ul>

==> The list items in <ul> can be represented with <li>

Syntax:

<ul>

<li> List Item-1 </li>

<li> List Item-2 </li>

</ul>

<!DOCTYPE html>

<html>

<head>

<title> Unordered List </title>

</head>

<body>

<h1> Java Fullstack </h1>

<ul>

<li> Core Java </li>

<li> HTML </li>

<li> CSS </li>

<li> JavaScript </li>

<li> Bootstrap </li>

<li> SQL </li>

<li> Oracle </li>

<li> Advanced Java </li>

<li> React JS </li>

<li> Springboot with Microservices </li>

</ul>

</body>

</html>

Nested List:

============

<!DOCTYPE html>

<html>

<head>

<title> Nesting of List </title>

</head>

<body>

<h1> Courses Offered </h1>

<ol>

<li> Java Fullstack

<ul>

<li> Database </li>

<li> Core Java </li>

<li> HTML-CSS-JavaScript-Bootstrap </li>

<li> Advanced Java </li>

<li> React JS </li>

<li> Springboot </li>

</ul>

</li>

<li> Python Fullstack

<ul>

<li> Core Python </li>

<li> Frontend Development </li>

<li> Advanced Python </li>

<li> Database </li>

<li> React JS </li>

<li> DJango for API Development </li>

</ul>

</li>

<li> Selenium Automation Testing </li>

<li> AWS Devops </li>

<li> Azure Devops </li>

<li> Generative AI </li>

</ol>

</body>

</html>

================================================

Navigation Page:

================

<!DOCTYPE html>

<html>

<head>

<title> Home </title>

</head>

<body>

<div>

<div>

<p>

For Enquiry: +91-8977029779

<a href = "#"> Our Free Tutorials </a>

</p>

</div>

<div>

<img src = "https://www.ashokitech.com/assets/images/logo2.jpg">

<div>

<ul>

<li> <a href = "#"> Home </a></li>

<li> <a href = "#"> Services </a></li>

<li> <a href = "#"> Courses Offered </a> </li>

<li> <a href = "#"> Contact </a> </li>

</ul>

</div>

</div>

</div>

</body>

</html>

**Day11: List Part\_02 and Tables:**

Description List/Definition List:

=================================

HTML:

Hyper Text Markup Language

Tag Based Language

required to display the content using tags on the browser

CSS:

Cascading Stylesheet

Property based language

can define the styles to the web content

JavaScript:

Statement based language

can define the functionality

tag ==> <dl> ==> description list tag

Syntax:

<dl> </dl> ==> can use to define the description list

<dt> ==> description term

Syntax:

<dt> Description Term </dt>

<dd> ==> Description Data

Syntax:

<dd> content/definition </dd>

Syntax for Description list:

<dl>

<dt> term-1: </dt>

<dd> definition </dd>

</dl>

===================================================

<!DOCTYPE html>

<html>

<head>

<title> Description List </title>

</head>

<body>

<h1> UI Terminology </h1>

<dl>

<dt> HTML: </dt>

<dd>

<ol>

<li> Hyper Text Markup Language </li>

<li> Tag Based Language </li>

<li> required to display the content using tags on the browser </li>

</ol>

</dd>

<dt> CSS: </dt>

<dd>

<ul>

<li> Cascading Stylesheet </li>

<li> Property based language </li>

<li> can define the styles to the web content </li>

</ul>

</dd>

<dt> JavaScript </dt>

<dd>

Statement based language<br>

can define the functionality

</dd>

</dl>

</body>

</html>

==========================================

normal representation:

<h1> to <h6>

<p>

<div>

Text Formats

Images

Links

List

Student data:

============

name1 ==> 77

name2 ==> 88

name3 ==> 68

Tables

======

==> table is the combination of rows and columns.

==> row ==> horizontal partition of the table

==> column ==> vertical partition of the row

<table>:

=======

Syntax:

<table> </table>

Note:

======

<table> tag cannot display the table with borders by default

<th> (table head tag):

=====================

Syntax:

<th> </th>

<tr> (table row)

================

Syntax:

<tr> </tr>

<td> (table data tag)

====================

Syntax:

<td> value </td>

<!DOCTYPE html>

<html>

    <head>

        <title> Tables </title>

    </head>

    <body>

        <table>

            <tr>

                <th> Id </th>

                <th> Name </th>

                <th> Course </th>

                <th> Marks </th>

                <th> Gender </th>

            </tr>

            <tr>

                <td> 1. </td>

                <td>Srikar</td>

                <td>JFSD</td>

                <td>98%</td>

                <td>M</td>

            </tr>

            <tr>

                <td> 2. </td>

                <td>Kalyan</td>

                <td>PFSD</td>

                <td>98%</td>

                <td>M</td>

            </tr>

            <tr>

                <td> 3. </td>

                <td>Saritha</td>

                <td>UI/UX</td>

                <td>98%</td>

                <td>F</td>

            </tr>

            <tr>

                <td> 4. </td>

                <td>Karthik</td>

                <td>QA</td>

                <td>97%</td>

                <td>M</td>

            </tr>

            <tr>

                <td> 5. </td>

                <td>Saroja</td>

                <td>JFSD</td>

                <td>98%</td>

                <td>F</td>

            </tr>

        </table>

    </body>

</html>

**Day12: Form Handling Part\_01:**

Form Handling:

==============

<form> tag:

===========

==> pair tag

Syntax:

<form> </form>

Input Section in the forms:

===========================

<input>

=======

=> to provide the input in the forms, we can use "<input>" tag

==> unpair element

==> self-closing tag

==> inline element.

Syntax:

<input>

type attribute:

===============

to define the input type

Syntax:

<input type = "value">

<label> tag:

============

for the form fields to provide/define any reference name we can define <label> tag

==> pair element

Syntax:

<label> reference name: </label>

==> inline element

Handling the Text box:

======================

type = "text"

Hidden the text box:

====================

type = "hidden"

Handling numbers:

=================

type = "number"

type = "tel"

Handling of mail id:

====================

type = "email"

Handling of button:

===================

type = "button"

Handling Date:

==============

type = "date"

Set the password:

=================

type = "password"

<!DOCTYPE html>

<html>

<head>

<title> Registration Form </title>

</head>

<body>

<div>

<img src = "https://www.ashokitech.com/assets/images/logo2.jpg">

</div>

<center>

<h1> Registration Form </h1>

</center>

<form>

<div>

<div>

<label> First Name: </label>

<input type="text"><br>

</div><br>

<div>

<label> Confirm First Name: </label>

<input type="text">

</div><br>

<div>

<label> Full Name: </label>

<input type = "hidden">

</div><br>

<div>

<label> Mobile Number: </label>

<input type = "number">

</div><br>

<div>

<label> Mobile Number: </label>

<input type="tel">

</div><br>

<div>

<label> Email: </label>

<input type="email">

</div><br>

<div>

<input type="button" value="submit">

</div><br>

<div>

<input type="submit" value = "Register">

</div><br>

<div>

<input type = "reset">

</div><br>

<div>

<input type = "button" value = "previous">

<input type = "button" value="next">

</div><br>

<div>

<label> Date Of Birth: </label>

<input type = "date">

</div><br>

<div>

<label> Password: </label>

<input type="password">

</div>

</div>

</form>

</body>

</html>

**Day13: Form Handling Part\_02:**

Form Handling

==============

<form>

<label>

type attribute

"text"

"email"

"password"

"number"

"tel"

"submit"

"reset"

"button"

"hidden"

value attribute

=========================================

type = "radio":

===============

ex:

gender:

male

female

other

<!DOCTYPE html>

<html>

<head>

<title> Forms </title>

</head>

<body>

<form>

<label> Gender: </label>

<input type = "radio" name = "gender"> <label> Male </label>

<input type = "radio" name = "gender"> <label> Female </label>

<input type = "radio" name = "gender"> <label> Other </label>

</form>

</body>

</html>

=================================

type = "datetime-local"

========================

type = "month"

==============

type = "range"

==============

type = "color"

==============

type = "time"

=============

type = "week"

=============

type = "file"

=============

type = "url"

============

type = "checkbox"

=================

<!DOCTYPE html>

<html>

<head>

<title> Forms </title>

</head>

<body>

<form>

<div>

<label> Gender: </label>

<input type = "radio" name = "g"> <label> Male </label>

<input type = "radio" name = "g"> <label> Female </label>

<input type = "radio" name = "g"> <label> Other </label>

</div><br>

<div>

<label> Time: </label>

<input type = "datetime-local">

</div><br>

<div>

<label> Month: </label>

<input type = "month">

</div><br>

<div>

<label> Rating the Faculty: </label>

<input type = "range" min = "1" max = "5">

</div><br>

<div>

<label> Chose the Color: </label>

<input type = "color">

</div><br>

<div>

<label> Time: </label>

<input type = "time">

</div><br>

<div>

<label> Week number: </label>

<input type = "week">

</div><br>

<div>

<label> Upload your CV: </label>

<input type = "file">

</div><br>

<div>

<label> Social Media Profiles: </label><br>

<input type = "url"> <label> Twitter: </label><br><br>

<input type = "url"><label> Linkedin: </label><br><br>

<input type = "url"><label> Facebook: </label><br>

</div><br>

<div>

<label> Select Your Hobbies: </label><br>

<input type = "checkbox"> <label> Reading Books </label>

<input type = "checkbox"> <label> Reading Novels </label>

<input type = "checkbox"> <label> Writing Stories </label>

<input type = "checkbox"> <label> Writing Songs </label>

<input type = "checkbox"> <label> Playing Online Games </label>

<input type = "checkbox"> <label> Playing Outdoor Games </label>

<input type = "checkbox"> <label> Watching Movies </label>

<input type = "checkbox"> <label> Chatting with Friends </label>

<input type = "checkbox"> <label> Surfing </label>

</div>

</form>

</body>

</html>

**Day14: Form Handling Part\_03:**

Form Handling:

==============

How to interlink label content with form element:

=================================================

for attribute ==> in <label>

id attribute ==> <input>

Syntax:

<label for = "id-value">

<input type = "value" id = "value">

<label> Gender: </label>

<label for = "male"> Male: </label> <input type = "radio" name = "g" id = "male"><br>

Female: <input type = "radio" name = "g" id = "female"><br>

Other: <input type = "radio" name = "g" id = "other">

===================================================

How to define the form elements with default values:

=====================================================

value attribute

===============

Syntax:

<input type = "value" value = "value">

<!DOCTYPE html>

<html>

<head>

<title> Forms </title>

</head>

<body>

<form>

<label for = "username"> User Name: </label>

<input type = "text" id = "username" value="username"><br><br>

<input type="submit">

</form>

</body>

</html>

user name: xxxxxx

password: xxxxxxxxxx

form validation:

================

1) we should create the server page

server-name.html

2) form.html ===> server.html

action attribute

<form>

Syntax:

<form action = "server\_page\_location">

form elements

</form>

3) whether the data after submission,

how it was reached to the server

method attribute

<form>

Syntax:

<form action = "location" method = "get/post">

</form>

<file:///C:/Users/Sai%20Srikar/Python-UI-Development/UI_Development_HTML_CSS/Day_11/formValidations/login.html>

<file:///C:/Users/Sai%20Srikar/Python-UI-Development/UI_Development_HTML_CSS/Day_11/login.html?username=kalyan.gudala%40gmail.com&password=password>

<!DOCTYPE html>

<html>

<head>

<title> Login Page </title>

</head>

<body>

<h1> Login Here </h1>

<form action="server.html" method="post">

<label for = "filed1"> User Name: </label>

<input type = "text" id = "filed1"name = "username" value = "abc@gmail.com"><br><br>

<label for = "field2"> Password: </label>

<input type="password" id = "field2" name = "password"><br><br>

<input type = "submit" value = "login">

</form>

</body>

</html>

===================================

Professional Form:

==================

<fieldset>

<legend> title for section </legend>

</fieldset>

<!DOCTYPE html>

<html>

<head>

<title> Professional Form </title>

</head>

<body>

<h1> Online Application Form </h1>

<form>

<fieldset>

<legend> Personal Information </legend>

<label> Name: </label>

<input type="text"><br>

<label> Confirm name: </label>

<input type="text">

</fieldset><br>

<fieldset>

<legend> Educational Details </legend>

<label> Graduation: </label>

<input type="text"><br>

<label> Intermediate (10 + 2): </label>

<input type="text"><br>

<label> SSC: </label>

<input type="text">

</fieldset>

<input type="submit">

</form>

</body>

</html>

**login.html:**

**===========**

<!DOCTYPE html>

<html>

    <head>

        <title> Form Submission </title>

    </head>

    <body>

        <h1> Login Here </h1>

        <form action ="server.html" method="get">

            <label for = "filed1"> User Name: </label>

            <input type = "text" id = "filed1"name = "username" value = "abc@gmail.com"><br><br>

            <label for = "field2"> Password: </label>

            <input type="password" id = "field2" name = "password"><br><br>

            <input type = "submit" value = "login">

        </form>

    </body>

</html>

**Server.html:**

**============**

<!DOCTYPE html>

<html>

    <head>

        <title> Form Submission Page </title>

    </head>

    <body>

        <h1> Submitted Successfully..!! </h1>

    </body>

</html>

**O/P:** [**file:///C:/Users/Sai%20Srikar/Python-UI-Development/UI\_Development\_HTML\_CSS/Day\_11/formValidation/server.html?username=kalyan%40gmail.com&password=password**](file:///C:/Users/Sai%20Srikar/Python-UI-Development/UI_Development_HTML_CSS/Day_11/formValidation/server.html?username=kalyan%40gmail.com&password=password)

**login.html:**

**===========**

<!DOCTYPE html>

<html>

    <head>

        <title> Form Submission </title>

    </head>

    <body>

        <h1> Login Here </h1>

        <form action ="server.html" method="post">

            <label for = "filed1"> User Name: </label>

            <input type = "text" id = "filed1"name = "username" value = "abc@gmail.com"><br><br>

            <label for = "field2"> Password: </label>

            <input type="password" id = "field2" name = "password"><br><br>

            <input type = "submit" value = "login">

        </form>

    </body>

</html>

**Server.html:**

**============**

<!DOCTYPE html>

<html>

    <head>

        <title> Form Submission Page </title>

    </head>

    <body>

        <h1> Submitted Successfully..!! </h1>

    </body>

</html>

**O/P:** [**file:///C:/Users/Sai%20Srikar/Python-UI-Development/UI\_Development\_HTML\_CSS/Day\_11/formValidation/server.html**](file:///C:/Users/Sai%20Srikar/Python-UI-Development/UI_Development_HTML_CSS/Day_11/formValidation/server.html)

**DAY15:** **Introduction To CSS:**

CSS INTRODUCTION:

=================

CSS ==> Cascading Style Sheets

One of the Language for UI development

to define the styles to the HTML content

Access HTML Elements

Features:

=========

1) Property based language

every property can define with value

Syntax:

property-name : value; (key : value)

2) not case sensitive language

CSS Versions:

=============

CSS-1 ==> Nov 1996

CSS-2 ==> Jul 1998

CSS-2.1 ==> Jun 2011

CSS-3 ==> 2013 Nov

=======================================

Ways to define CSS:

===================

==> 3-ways:

1) Inline Stylesheet

2) Internal Stylesheet

3) External Stylesheet

1) Inline Stylesheet

=====================

==> can use to define the style for the only one particular HTML element at a time.

==> style attribute

Syntax:

<tag style = "property-name : value; property-name : value;"> Content </tag>

<!DOCTYPE html>

<html>

<head>

<title> Inline CSS </title>

</head>

<body>

<h1 style="text-align: center;color: blue;"> UI Development </h1>

<h2 style="color: violet;"> HTML </h2>

<h3 style="color: skyblue;"> Tag Names </h3>

<h3 style="color: skyblue;"> Attributes </h3>

<h2 style="color: violet;"> CSS </h2>

<h3 style="color: skyblue;"> Selectors </h3>

<h3 style="color: skyblue;"> Properties </h3>

<h2 style="color: violet;"> JavaScript </h2>

<h3 style="color: skyblue;"> Statements </h3>

</body>

</html>

Disadvantage:

=============

1) Using the Inline CSS, we can define the style for only one element at a time.

2) To add the same style among multiple elements, we should re-write the same code.

=================================

2) Internal Stylesheet:

=======================

==> to define the block of style which are commonly defined among more than one element at a time without re-writing, we can use "Internal Stylesheet".

==> To define the Internal stylesheet:

we can use "<style>" tag

in <head> tag.

Syntax:

<head>

<style>

selector{

property-name : value;

property-name : value;

}

</style>

</head>

<!DOCTYPE html>

<html>

<head>

<title> Internal Stylesheet </title>

<style>

p{

color: blueviolet;

text-align: center;

}

</style>

</head>

<body>

<h1> UI Development </h1>

<div>

<h2> HTML </h2>

<p>

HTML Includes: <br>

Tags ==> some of tags are paired and some of tags are Unpaired.<br>

Attributes

</p>

</div>

<div>

<h2> CSS </h2>

<p>

CSS Includes:<br>

Selectors for access the HTML Elements <br>

Properties for defining the styles.

</p>

</div>

<div>

<h2> JavaScript </h2>

<p>

JavaScript contains:<br>

Statements with tokens<br>

To define the functionality on webpage.

</p>

</div>

</body>

</html>

Disadvantage:

=============

==> Using the internal stylesheet,

we can define the style within the same webpage.

==> to add the same style of one webpage among multiple pages of same website,

Internal stylesheet can required the re-writing of the code.

=========================================

3) External Stylesheet:

=======================

Website ==> multiple webpages

==> we need to define the styles externally in ".css" file and apply those external styles to our webpages using "<link>" tag.

1) create the css file

Syntax:

file-name.css

2) write the styles into .css file

3) link that css file html file

<head>

<link rel = "stylesheet" href = "file-location">

</head>

<!DOCTYPE html>

<html>

<head>

<title> External Stylesheet </title>

<link rel = "Stylesheet" href="styles.css">

</head>

<body>

<h1> Ashok IT </h1>

</body>

</html>

**Day16: Basic CSS Selectors:**

**Basic CSS Selectors:**

======================

==> Selector: can use to access the HTML elements

==> Three basic selectors:

1) Tag Selector

2) Class Selector

3) Id Selector

**1) Tag Selector**

===============

==> can use with "tag name"

Syntax:

tag-name{

property1 : value;

property2 : value;

}

**2) Id Selector:**

===============

Syntax:

#Id-value{

property1 : value;

property2 : value;

}

**3) Class Selector:**

==================

Syntax:

.class-name{

property1 : value;

property2 : value;

}

<!DOCTYPE html>

<html>

<head>

<title> Tag Based Selector </title>

<style>

h1{

text-align: center;

color: rgb(15, 83, 104);

background-color: azure;

}

#Section1{

border: 3px solid royalblue;

}

#Section2{

border: 3px dashed rosybrown;

}

.one{

color: turquoise;

}

.two{

color: blueviolet;

}

p{

text-align: center;

background-color: aquamarine;

color: burlywood;

}

</style>

</head>

<body>

<h1> Ashok IT </h1>

<div id = "Section1">

<h2 class = "one"> Java Fullstack </h2>

<h3 class = "two"> Content </h3>

<p>

UI Development <br>

Backend Development<br>

Database <br>

Devops Tools

</p>

</div><br>

<div id = "Section2">

<h2 class = "two"> Python Fullstack </h2>

<h3 class = "one"> Content </h3>

<p>

UI Development <br>

Backend Development<br>

Database <br>

Devops Tools

</p>

</div>

</body>

</html>

**Day17: Font Properties & Text Properties:**

CSS PROPERTIES:

===============

Font Properties:

================

To handle the font for the text, we can use "font-properties"

1) font-family:

===============

default font: Times New Roman

Syntax:

font-family : value;

Note:

====

if the value with one word ==> value without any quotes

if the value with more than one word ==> value with quotes

==> font-family property can allow to define with single value

Syntax:

font-family : "value" or value;

==> font-family can allow to define with more than one value

Syntax:

font-family : "value1","value2","value3",...;

==> In this case, the font-family property can check the values from left towards right.

That means, it can check the left most value if it supported by our browser then it can apply. Otherwise, the next value can check and apply it is supporting.

If any value is not supported automatically, the font can define with "Times New Roman".

===========================================================

2) font-size:

==============

==> property can be used to define the size to the font

Syntax:

font-size : number;

Unit: px/%/em/vh/vw

default size of any font: 16px

====================================================

3) font-weight:

===============

==> can use to define the font with either bold or thin

Syntax:

font-weight : normal | bold | bolder | lighter | number;

Note:

=====

font-weight : number;

In this, no need to use/represent any unit.

Automatically, it can detect in px.

===================================================

4) font-style:

==============

==> can use to define the italic nature to the font.

Syntax:

font-style : normal | italic;

5) font-variant:

================

small caps ==> in the given word, all the letters should be in upper case. and the first letter of each word in larger in size remaining all in smaller in size.

==> use to apply the small caps to the font

Syntax:

font-variant : normal | smallcaps;

===============================================

6) font shorthand property:

===========================

using this property, we can define all other font properties at a time.

Syntax:

font : font-style font-variant font-weight font-size font-family;

<!DOCTYPE html>

<html>

<head>

<title> Font Properties </title>

<style>

/\* body{

font-family:Verdana;

} \*/

\*{

font-family: 'Courier New', Courier, monospace;

font-style: oblique;

}

p{

font-size: 120%;

font-weight: bold;

}

h1{

/\* font-weight: 1400;

font-variant: small-caps; \*/

font : italic small-caps normal 400% Verdana;

}

</style>

</head>

<body>

<h1> Ashok IT </h1>

<div>

<h2> Java Fullstack </h2>

<p>

Web Technologies: HTML, CSS, JavaScript, Bootstrap <br>

Web Frameworks : React JS/ Angular JS<br>

Core Java<br>

Advanced Java<br>

Database Programming<br>

Framework: Springboot

</p>

</div>

<div>

<h2> Python Fullstack </h2>

<p>

Web Technologies: HTML, CSS, JavaScript, Bootstrap <br>

Web Frameworks : React JS/ Angular JS<br>

core Python<br>

Advanced Python <br>

Python Libraries <br>

Database Programming<br>

Framework: DJango

</p>

</div>

</body>

</html>

==============================================================

Text Properties:

================

1) Text-align:

==============

==> to define the alignment to the text (left alignment/right alignment/center alignment)

Syntax:

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

=================================================

2) color:

==========

==> used to apply the color to the text.

default color: black

Syntax:

color : value;

here:

value ==> color-name | hexa code | rgb() | rgba()

hexa code:

==========

red, green and blue

Syntax:

#rrggbb

Range: red/green/blue ==> 0 to 9 and a to f

ex: #f12398

rgb()

=====

rgb(value for red, value for green, value for blue)

range: 0 to 255

rgba(red, green, blue, alpha)

opacity ==> 0 to 1 ==> transparency to the text

<!DOCTYPE html>

<html>

<head>

<title> Font Properties </title>

<style>

/\* body{

font-family:Verdana;

} \*/

\*{

font-family: 'Courier New', Courier, monospace;

font-style: oblique;

}

p{

font-size: 120%;

font-weight: bold;

color: rgba(124, 218, 231,1.0);

}

h1{

/\* font-weight: 1400;

font-variant: small-caps; \*/

font : italic small-caps normal 400% Verdana;

text-align: center;

color: darkblue;

}

h2{

color:#987654;

}

</style>

</head>

<body>

<h1> Ashok IT </h1>

<div>

<h2> Java Fullstack </h2>

<p>

Web Technologies: HTML, CSS, JavaScript, Bootstrap <br>

Web Frameworks : React JS/ Angular JS<br>

Core Java<br>

Advanced Java<br>

Database Programming<br>

Framework: Springboot

</p>

</div>

<div>

<h2> Python Fullstack </h2>

<p>

Web Technologies: HTML, CSS, JavaScript, Bootstrap <br>

Web Frameworks : React JS/ Angular JS<br>

core Python<br>

Advanced Python <br>

Python Libraries <br>

Database Programming<br>

Framework: DJango

</p>

</div>

</body>

</html>

**Day18: Text Properties:**

background-color:

=================

==> to set the background color to the text/block, background-color property can be used.

Syntax:

background-color : color-name | hexa-code | rgb() | rgba()

==============================

text-decoration:

================

==> to define the decorations like:

underline

overline

line-through

"text-decoration" property can be used.

Syntax:

text-decoration : text-decoration-type text-decoration-color text-decoration-style;

here:

1) text-decoration-type : underline | overline | linethrough;

2) text-decoration-color

3) text-decoration-style : solid | dotted | dashed | double | wavy;

=====================================

text-transform:

===============

==> to define the case of the text we can use "text transform" property.

Syntax:

text-transform : none | uppercase | lowercase | capitalize case

================================

text-indent:

============

==> the space for the first line of the paragraph at the beginning.

Syntax:

text-indent : number;

===============================================

letter-spacing:

===============

==> to define the space between the letters "letter-spacing" can be used.

Syntax:

letter-spacing : number;

================================================

word-spacing:

=============

Syntax:

word-spacing : number;

=============================

line-height:

============

==> use define the space between lines

Syntax:

line-height : number;

===========================================

direction:

==========

==> can use to define the text direction to display.

Syntax:

direction : ltr | rtl;

==========================================================

text-shadow:

============

==> to define the shadow to the text

Syntax:

text-shadow : h-shadow v-shadow blur-shadow color;

==============================================

.html code:

===========

<!DOCTYPE html>

<html>

<head>

<title>

Text Properties

</title>

<link rel = "stylesheet" href = "styles.css">

</head>

<body>

<div>

<div> <h2> The Bright Place to Build The Best Career Value </h2>

<p> Best Opportunity To Learn From Currently Working Professionals </p>

</div>

<div id = "sec1">

Online Training

</div><br>

<div id = "sec2"> Classroom Training </div><br>

<div id = "sec3"> Corporate Training </div><br>

<div id = "sec4"> Weekend Workshops </div><br>

<div id = "sec5"> Placement </div><br>

<div id = "sec6"> Internships </div>

</div>

<div>

<p>

I have completed multiple certification courses from Ashok IT including Java, cloud computing, and data structures and algorithm. I was greatly impressed by the commendable teaching methodologies and experienced and insightful tutors who were able to simplify the process of learning various complex technologies

</p>

</div>

</body>

</html>

============================

.css file:

==========

#sec1{

width: 200px;

height: 30px;

border: 3px solid black;

background-color: green;

color: white;

text-align: center;

text-transform: uppercase;

}

#sec2{

width: 200px;

height: 30px;

border: 3px solid black;

background-color: blue;

color: white;

text-align: center;

text-transform: uppercase;

}

#sec3{

width: 200px;

height: 30px;

border: 3px solid black;

background-color: orange;

color: white;

text-align: center;

text-transform: uppercase;

}

#sec4{

width: 200px;

height: 30px;

border: 3px solid black;

background-color: purple;

color: white;

text-align: center;

text-transform: uppercase;

}

#sec5{

width: 200px;

height: 30px;

border: 3px solid black;

background-color: red;

color: white;

text-align: center;

text-transform: uppercase;

}

#sec6{

width: 200px;

height: 30px;

border: 3px solid black;

background-color: rgb(115, 152, 204);

color: white;

text-align: center;

text-transform: uppercase;

}

h2{

text-decoration: underline rebeccapurple dotted;

text-transform: capitalize;

/\* direction: rtl; \*/

text-align: center;

text-shadow: -10px -15px 7px #5ac4d7;

}

p{

text-indent: 25px;

letter-spacing: 2px;

word-spacing: 10px;

line-height: 20px;

}

**DAY19**: **Box Model Part\_01:**

Box Model:

==========

Box consisting of:

1) Margin ==> can be used to separate html elements within the web page.

2) Content

3) Padding ==> Outer layer to the content

The separation between the content and border

4) Border ==> the separation between margin and padding

Content Properties:

===================

==> several properties:

1) width

2) height

3) float

width:

=====

==> property can be used to define the horizontal size

Syntax:

width : number;

height:

========

==> can use the representation of vertical size

Syntax:

height : number;

float:

======

==> use to represent the content of the block container side by side

Syntax:

float : left | right;

Margin Properties:

==================

1) margin:

==========

to define the margin in all four sides (top, right, bottom and left), we can use "margin" property.

syntax:

margin : number; // can apply the margin in all four sides with same value

margin : n1 n2; // n1 ==> apply top and bottom and n2 ==> right and left

margin : n1 n2 n3; // n1 ==> top, n2 ==> right, n3 ==> bottom and n2 ==> left

margin : top right bottom left;

2) margin-top

=============

==> can use to define the margin in only top side of the box

Syntax:

margin-top : number;

3) margin-right

===============

==> can use to define the margin in right of the box.

Syntax:

margin-right : number;

4) margin-bottom

=================

can use to define the margin in bottom of the box.

Syntax:

margin-bottom : number;

5) margin-left

==============

can use to define the margin in left of the box.

Syntax:

margin-left : number;

===============================================

Padding Properties:

===================

==> to differentiate the content and the border, padding properties can be used.

1) padding:

===========

Syntax:

padding : number; // to apply the padding in all four sides with the same value

padding : n1 n2; // n1 ==> top and bottom / n2 ==> right and left

padding : n1 n2 n3; // n1 ==> top, n2 ==> right, n3 ==> bottom and n2 ==> left

padding : top right bottom left;

2) padding-top

3) padding-right

4) padding-bottom

5) padding-left

=============================================================

<!DOCTYPE html>

<html>

<head>

<title> India And States </title>

<link href="style.css" rel="stylesheet">

</head>

<body>

<h1> India - States </h1>

<div class = "india">

<div class = "state" id = "s01"> Andhrapradesh </div>

<div class = "state" id = "s02"> Arunachalpradesh </div>

<div class="state" id = "s03"> Telangana </div>

<div class="state" id = "s04"> Karnataka </div>

<div class="state"> Kerala </div>

<div class="state"> Chennai </div>

<div class="state"> Maharashtra </div>

<div class="state"> Delhi </div>

<div class="state"> Assam </div>

<div class="state"> WestBengal</div>

</div>

</body>

</html>

==================================

.css file:

==========

.state{

width: 150px;

height: 60px;

/\* margin: 20px; \*/

/\* margin-left: 30px;

margin-top: 100px;

margin-right: 0px;

margin-bottom: 60px; \*/

/\* margin : 40px 60px; \*/

/\* margin: 40px 60px 80px; \*/

margin : 40px 60px 80px 100px;

border : 2px solid red;

/\* padding: 40px; \*/

/\* padding-top: 30px;

padding-right: 0px;

padding-bottom: 0px;

padding-left: 0px; \*/

padding : 30px 0px 0px 0px;

text-align: center;

}

**Day20: Box Model Part\_02:**

Box Model:

==========

float property:

===============

<p>, <div> etc. => Block elements

==> float property is a content property

can use to display multiple block elements side by side (with in a line)

Syntax:

float : left | right;

==============================================

Border Properties:

==================

1) border-width : number;

==> used to define the thickness to the border

2) border-style : none | solid | dotted | dashed | double | groove | inset | outset;

==> used to define the type of the border.

3) border-color : color-name | hexa | rgb;

==> used to define the color to the border.

4) border: width style color;

==> shorthand property

5) border-top : width style color;

6) border-right : width style color;

7) border-bottom : width style color;

8) border-left : width style color;

9) border-radius : number;

10) box-shadow : h-offset v-offset blur spread color;

11) outline-offset: number;

**Day21: List and Link Styles:**

LIST STYLES

===========

ORDERED LIST:<OL> AND <LI>

LIST ITEMS ==> 1 2 3...

UNORDERED LIST : <UL> AND <LI>

LIST ITEMS ==> DISC

DEFINITION LIST: <DL>, <DT> AND <DD>

TERM1:

DESCRIPTION

TERM2:

DESCRIPTION

NESTED LIST

list-style-type:

=================

==> for <ol> and <ul> tag

Syntax:

list-style-type: disc | square | circle |

decimal | decimal-leading-zero | lower-alpha | upper-alpha |

lower-roman | upper-roman | none

Note:

====

HTML has two types of attributes:

1) Valued attributes

Syntax:

attribute-name = "value"

ex: start = "100"

style = "color:red;"

2) Self-valued attributes

Syntax:

attribute-name = "attribute-name"

or

attribute-name

ex: reversed = "reversed"

start attribute:

================

==> for ordered list,

to specify the starting point value "start" attribute can be used.

Syntax:

<ol start = "value"> </ol>

here:

value = "decimal"

can be applicable for both decimal numbering and other numbering etc.

reversed attribute:

===================

==> self-valued attribute

==> to number the ordered list in decreasing order, "reversed" attribute can be used.

Syntax:

<ol reversed></ol>

<!DOCTYPE html>

<html>

<head>

<title> Ordered List </title>

</head>

<body>

<h1> Java Fullstack Development </h1>

<ol start="101" reversed>

<li> Core Java </li>

<li> Frontend Technologies </li>

<li> Database </li>

<li> Advanced Java </li>

<li> Springboot Microservices </li>

<li> Angular/React JavaScript </li>

<li> Real-Time Project Oriented Training </li>

<li> Cloud Technologies </li>

</ol>

</body>

</html>

==============================================

<!DOCTYPE html>

<html>

<head>

<title> Ordered List </title>

<style>

.ordered{

list-style-type: lower-roman;

}

</style>

</head>

<body>

<h1> Java Fullstack Development </h1>

<ol class = "ordered">

<li> Core Java </li>

<li> Frontend Technologies </li>

<li> Database </li>

<li> Advanced Java </li>

<li> Springboot Microservices </li>

<li> Angular/React JavaScript </li>

<li> Real-Time Project Oriented Training </li>

<li> Cloud Technologies </li>

</ol>

</body>

</html>

==========================================

Representing the list items with symbols:

=========================================

list-style-image:

=================

==> applicable for <ul> tag.

Syntax:

list-style-image : url("image path");

<!DOCTYPE html>

<html>

<head>

<title> Ordered List </title>

<style>

.unordered{

list-style-image: url("star1.png");

}

</style>

</head>

<body>

<h1> Java Fullstack Development </h1>

<ul class = "unordered">

<li> Core Java </li>

<li> Frontend Technologies </li>

<li> Database </li>

<li> Advanced Java </li>

<li> Springboot Microservices </li>

<li> Angular/React JavaScript </li>

<li> Real-Time Project Oriented Training </li>

<li> Cloud Technologies </li>

</ul>

</body>

</html>

=================================================

Link Styles:

============

==> 4-different link states:

1) Unvisited Link ==> link state ==> a : link

2) Visited link ==> a : visited

3) Active link ==> a : active

4) Hover Link ==> a : hover

<!DOCTYPE html>

<html>

<head>

<title> Link Styles </title>

<style>

a:link{

color: cadetblue;

text-decoration: none;

}

a:visited{

color: chocolate;

}

a:active{

color: red;

}

a:hover{

background-color: aqua;

text-decoration: underline;

}

</style>

</head>

<body>

<div class = "linkstates">

<div>

<a href = "https://www.apollohospitals.com/"> Link1 </a>

</div>

<div>

<a href = "https://www.flipkart.com/"> Link2 </a>

</div>

<div>

<a href = "https://us06web.zoom.us/meeting#/upcoming"> Link3 </a>

</div>

</div>

</body>

</html>

**Day22: Navigation Bar:**

Navigation Bar:

===============

Prerequisites:

div tag

<a> tag

ul tag

CSS Poperties

box-sizing:

===========

box ==> margin, border, padding and content (width and height)

==> box-sizing property can be used to specify whether we want include the width and height box or not

box-sizing : content-box | border-box

<!DOCTYPE html>

<html>

<head>

<title> Home Page </title>

<style>

\*{

box-sizing: border-box;

}

#navbar{

/\* background-color: lightcyan; \*/

height: 70px;

}

#navbar .navbar-brand{

float: left;

background-color: blueviolet;

width: 20%;

padding-top: 10px;

padding-left: 50px;

padding-bottom: 10px;

}

#navbar .nav{

float: left;

background-color: red;

width: 80%;

}

#navbar .nav ul{

list-style-type: none;

padding-bottom: 10px;

}

#navbar .nav ul li{

float: left;

padding-left: 20px;

padding-right: 30px;

}

#navbar .nav ul li a:link{

text-decoration: none;

color: white;

}

#navbar .nav ul li a:hover{

background-color: white;

color: red;

}

</style>

</head>

<body>

<h1> My Tour Package </h1>

<div id = "navbar">

<div class = "navbar-brand"> My Travel Page </div>

<div class = "nav">

<ul>

<li> <a href="#"> Home </a></li>

<li> <a href = "#"> Packages </a> </li>

<li> <a href="#"> Climate </a></li>

<li> <a href = "#"> Contact </a></li>

</ul>

</div>

</div>

</body>

</html>

**Day23: Image Gallery, Background Styles:**

Image Gallery:

==============

==> Collection of images,

that each image can define with hover effect.

pre-requisites:

div tag

img tag

<!DOCTYPE html>

<html>

<head>

<title> Image Gallery </title>

<style>

.photo-gallery{

width: 95%;

border: 1px solid blue;

margin: auto;

margin-top: 100px;

padding: 10px;

height: 400px;

}

.photo{

float: left;

margin: 10px;

cursor: pointer;

}

.photo img{

border: 2px ridge red;

transition: transform 0.5s;

}

.photo div{

text-align: center;

}

.photo img:hover{

transform: scale(-1.1);

/\* transform: rotate(180deg); \*/

}

</style>

</head>

<body>

<div class = "photo-gallery">

<div class = "photo">

<img src = "Java Fullstack.jpg" height="200" width="200">

<div> Java Fullstack </div>

</div>

<div class = "photo">

<img src = "Software Testing.jpg" height="200" width="200">

<div> Software Testing </div>

</div>

<div class = "photo">

<img src = "Spring boot.jpg" height="200" width="200">

<div> Springboot Microservices </div>

</div>

</div>

</body>

</html>

======================================================

transform:

==========

==> it is a CSS property

can use to scale or rotate an element.

Syntax:

transform : scale(value);

transform : rotate(value-deg);

transition:

===========

==> it is a CSS property

can be used to apply the transformation linearly/gradually.

Syntax:

transition : transform time-in-sec;

==================================================

Background styles:

==================

1) background-image

====================

==> to set the image as the background, we can use "background-image" property.

Syntax:

background-image : url("file location);

2) background-repeat:

======================

when the background-image resolution is smaller than our screen resolution,

then: the image can be repeated for several times.

==> to specify whether the background image need to be repeated or not we have to use "background-repeat" property.

Syntax:

background-repeat : repeat | no-repeat | repeat-x | repeat-y;

3) background-size:

===================

if the background-image with no-repeat, then:

background-size can be defined.

Syntax:

background-size : auto height width flex;

4) background-position:

=======================

==> to set the position for the background-image, we have to use "background-position".

Syntax:

background-position : top | left top | right top | center top |

center | left center | right center | center center |

bottom | left bottom | right bottom | center bottom;

<!DOCTYPE html>

<html>

<head>

<title> Image Gallery </title>

<style>

body{

background-image: url("Spring\ boot.jpg");

background-repeat: no-repeat;

background-size: 50% 50%;

background-position: center bottom;

}

.photo-gallery{

width: 95%;

border: 1px solid blue;

margin: auto;

margin-top: 100px;

padding: 10px;

height: 400px;

}

.photo{

float: left;

margin: 10px;

cursor: pointer;

}

.photo img{

border: 2px ridge red;

transition: transform 0.5s;

}

.photo div{

text-align: center;

}

.photo img:hover{

transform: scale(-1.1);

/\* transform: rotate(180deg); \*/

}

</style>

</head>

<body>

<div class = "photo-gallery">

<div class = "photo">

<img src = "Java Fullstack.jpg" height="200" width="200">

<div> Java Fullstack </div>

</div>

<div class = "photo">

<img src = "Software Testing.jpg" height="200" width="200">

<div> Software Testing </div>

</div>

<div class = "photo">

<img src = "Spring boot.jpg" height="200" width="200">

<div> Springboot Microservices </div>

</div>

</div>

</body>

</html>

**Day24: Multimedia Handling:**

Multiple Background Images:

===========================

<!DOCTYPE html>

<html>

<head>

<title> Multiple Backgrounds </title>

<style>

body{

background-image: url("Image01.jpg"),url("Image02.jpg");

background-size: 50% 300px;

background-repeat: no-repeat;

background-position: left top;

}

</style>

</head>

<body>

</body>

</html>

==================================================

<figure> tag:

=============

Syntax:

<figure>

<img src = "image location" height = "px" width = "px" alt = "text">

<figcaption> Caption <figcaption>

</figure>

<!DOCTYPE html>

<html>

<head>

<title> Figure Tag</title>

</head>

<body>

<figure>

<img src="Image02.jpg" height="400px" width="400px" alt="Image Is not loading">

<figcaption> fig: Image with Nature </figcaption>

<img src="Image01.jpg" height="400px" width="400px" alt = "Image is not loading">

<figcaption> Random Image </figcaption>

</figure>

<img src="Image02.jpg" height="400px" width="400px" alt="Image Is not loading">

<img src="Image01.jpg" height="400px" width="400px" alt = "Image is not loading">

</body>

</html>

========================================

<audio> Tag:

============

Syntax:

<audio>

<source src = "file.mp3">

<source src = "file.ogg">

</audio>

<!DOCTYPE html>

<html>

<head>

<title> Audio Tag </title>

</head>

<body>

<audio controls muted loop autoplay>

<source src = "song.mp3">

<source src = "song.ogg">

</audio>

<audio controls muted>

<source src = "song.mp3">

<source src = "song.ogg">

</audio>

</body>

</html>

==========================================

<video> tag:

============

<!DOCTYPE html>

<html>

<body>

<video height="400px" width="400px" muted loop autoplay controls>

<source src="Video01.mp4">

</video>

</body>

</html>

==================================================

Display Youtube Video on web page:

==================================

<!DOCTYPE html>

<html>

<body>

<iframe width="560" height="315" src="https://www.youtube.com/embed/Ri-URt8gPCk?si=Y8LG8MKZUKmYIeAG" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" referrerpolicy="strict-origin-when-cross-origin" allowfullscreen></iframe>

<iframe width="560" height="315" src="https://www.youtube.com/embed/qJ8gUp0O25k?si=CTIGdBkH7DZOMyTp" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" referrerpolicy="strict-origin-when-cross-origin" allowfullscreen></iframe>

</body>

</html>

**Day25: Tables with CSS:**

Professional with striped Table:

================================

<!DOCTYPE html>

<html>

<head>

<title> Professional Table </title>

<style>

.table{

width: 100%;

/\* background-color: tomato; \*/

cursor: pointer;

border-collapse: collapse;

margin-top: 10px;

margin-bottom: 30px;

}

.table-container{

width: 95%;

margin:auto;

}

.table td{

border: 1px solid blue;

padding: 8px;

text-align: center;

}

.table th{

border: 1px solid red;

padding: 12px 10px;

/\* text-align: left; \*/

background-color: rgba(128, 128, 128, 0.354);

/\* color: white; \*/

}

.table tr:nth-child(even){

background-color: rgba(173, 216, 230, 0.386);

}

.table tr:nth-child(odd){

background-color: white;

}

</style>

</head>

<body>

<h1> States- Capital and Language </h1>

<div class = "table-container">

<table class = "table">

<thead>

<tr>

<th> S.No. </th>

<th> State Name </th>

<th> Capital Name </th>

<th> Language </th>

</tr>

</thead>

<tbody>

<tr>

<td> 1. </td>

<td> Andhra Pradesh </td>

<td> Amaravathi </td>

<td> Telugu </td>

</tr>

<tr>

<td> 2. </td>

<td> Telangana </td>

<td> Hyderabad </td>

<td> Telugu </td>

</tr>

<tr>

<td> 3. </td>

<td> Arunachal Pradesh </td>

<td> Itanagar </td>

<td> English </td>

</tr>

<tr>

<td> 4. </td>

<td> Kerala </td>

<td> Thiruvanthapuram </td>

<td> Malayalam </td>

</tr>

</tbody>

</table>

</div>

</body>

</html>

**Day26: Page Layout:**

TWO TYPES:

1) STATIC PAGES : HTML, CSS, JS

2) RESPONSIVE PAGES : HTML, CSS, JS, Bootstrap

HTML Layout:

============

1) Header Section:

==================

==> used to specify the external information/additional information to our development.

external information like:

about author/developer

location

team size

==> <header> </header>

2) Navigation Section:

======================

<div>

<a>

<ul>

<nav> </nav>

3) Index Section:

=================

==> optional

to make define/post any advertisements

4) content Section:

===================

<main> </main>

5) Footer Section:

==================

<footer> </footer>

<!DOCTYPE html>

<html lang="en">

<html>

<header>

<head>

<title> WebPage Layout </title>

<style>

h1{

color: red;

}

</style>

</head>

</header>

<body>

<h1> Ashok IT </h1>

<nav>

<a href = "#"> Home </a>

<a href = "#"> Contact </a>

</nav>

<main>

<div class = "section1">

<p></p>

<h2></h2>

<img>

</div>

</main>

<footer>

<address>

<h1> Ashok IT </h1>

<label> Address: </label>

Swathi Anukar Building,<br>

Floor: 5th floor,<br>

Block: 2nd block,<br>

Ph:<br>

email:<a href = "mailto : ravivraoinfs@gmail.com"> ravivraoinfs@gmail.com </a>

</address>

</footer>

</body>

</html>

</html>

**Day27: Introduction To JavaScript:**

Introduction To JavaScript

==========================

==> to add the functionality to our web elements, we should use the javascript.

what is JavaScript?

====================

==> powerful, high-level and scripting language.

JavaScript can be used with:

1) in HTML ==> Client side

2) Node.Js ==> Server side

3) Mongo DB ==> Database

4) 3DS Max, Revit ==> Animations

JavaScript ==>

two different Compilation Techniques:

JIT ==> Just In Time

AOT ==> Ahead Of Time

JS can run ==> browser

JS can run ==> application

is JS compiler based or interpreter based?

============================================

V8, babel, typescript et

History of JavaScript:

======================

Mosaic ==> browser

Markup Languages: GML, SGML, HTML

Scripting language: ECMAScript

1994,

Netscape Corporation

Netscape browser

Brenden Eich ==> written a language "Mocha"

Mocha ==> renamed by Netscape ==> Live Script

1998

Microsoft ==> launched an OS ==> win98

free browsers: IE (Internet Explorer), Chrome etc.

Live Script ==> Sun Micro System

Live Script ==> JavaScript

Netscape ==> ECMA

JavaScript ==> ECMASCRIPT

Ecma Script ==> standard

JavaScript ==> practical implementation

===============================================

Why the JS?

===========

DOM (Document Object Model) Operations

=======================================

1) Adding of elements

2) Removing of Elements

3) Replace etc.

BOM (Browser Object Model) Operations

======================================

Location

History

Navigations

Popups/Alerts etc.

=====================================================

How to use the JS?

==================

three ways:

1) Inline JS

2) Embedded JS

3) External JS

**Day28: Handling of HTML Elements using JS:**

JAVASCRIPT:

CAN HANDLE HTML ELEMENTS

CONSISTING OF THREE THINGS:

1) PROPERTIES

2) FUNCTIONS/METHODS

3) EVENTS

BALL

DIAMETER - 5MM

COLOR - RED

shape - spherical

fly

crawl

kick ball

push ball

void m1()

{

}

class myClass{

void m1()

{

}

}

HTML:

<h1 style = "color:red"> </h1>

HOW TO WRITE JAVASCRIPT?

========================

==> three ways:

1) Inline JavaScript

2) Embedded JavaScript/Internal JavaScript

3) External JavaScript

1) Inline JavaScript

====================

==> WHEN WE WANT TO DEFINE THE FUNCTIONALITY FOR ONLY ONE ELEMENT WITHIN THE WEBPAGE, WE CAN USE "INLINE JAVASCRIPT".

==> DRAWBACK:

WE CAN RE-USE THE SAME SCRIPT WHEN WE NEED TO DEFINE FOR MULTIPLE ELEMENTS WITHIN THE SAME PAGE.

==> TO OVERCOME THIS, WE CAN USE "EMBEDDED JAVASCRIPT".

<!DOCTYPE html>

<html lang = "en">

<head>

<title> Inline JavaScript </title>

</head>

<body>

<button onclick="window.print()"> Print </button>

</body>

</html>

2) EMBEDDED JAVASCRIPT:

=======================

==> ALSO CALLED AS "INTERNAL JAVASCRIPT".

==> WE CAN USE: "<SCRIPT>".

<!DOCTYPE html>

<html lang = "en">

<head>

<title> Embedded JavaScript </title>

</head>

<body>

<script>

a = 100

b = 200

document.write("The Sum = "+(a+b));

console.log("The Sum = "+(a+b));

</script>

</body>

</html>

3) External JavaScript:

=======================

<script src = "javaScript file"> </script>

Extension JavaScript file:

file-name.js

<!DOCTYPE html>

<html lang = "en">

<head>

<title> Embedded JavaScript </title>

</head>

<body>

<script src = "script.js">

</script>

</body>

</html>

HOW THE JAVASCRIPT CAN HANDLE THE HTML ELEMENTS?

================================================

<!DOCTYPE html>

<html>

<head>

<title>

Accessing of HTML Elements

</title>

<script>

function FillingContent()

{

document.images[0].src = "Bike.jpg";

document.forms[0].elements[1].value = "Register";

document.forms[1].elements[1].value = "Login";

}

</script>

</head>

<body onload="FillingContent()">

<h1> Accessing of HTML Elements using JavaScript </h1>

<img height="100" width="100" border = "1">

<div>

<form>

User Name: <input type="text">

<input type = "button">

</form>

</div><br>

<div>

<form>

Email: <input type="email">

<input type = "button">

</form>

</div>

</body>

</html>

**Day29: Identifiers, Variables with var keyword:**

IDENTIFIERS

===========

methods, variables, classes etc.

==> within the program, to name any entity like:

variables, methods, classes, functions, objects etc.

we can use "identifiers"

Rules for the Identifiers:

==========================

1) Should include:

Alphabets ==> Upper case and/or Lower case

Digits ==> 0 to 9

Special Characters:

Underscore

Dollar sign

ex: deposit\_12, deposit$21

2) Identifier never start with digit.

ex: 9abc = 12;

3) Identifier never be the keyword.

4) Never include with special characters like:

space, dot, <, >, ?/"{

5) No limit in identifier length.

Note: When we need to define identifiers, the length is as per our standards of working project.

<!DOCTYPE html>

<html>

<script>

abc = 10; // valid identifier

a\_12 = 20; // valid identifier

a$21 = 30; // valid identifier

// a.c = 30;

// 9abc = 32;

// else = 321;

// a<b = 123;

a = 21;

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa = 100;

document.write(abc+"<br>");

document.write(a\_12+"<br>");

document.write(a$21+"<br>");

// document.write(a.c);

// document.write(9abc);

document.write(a+"<br>");

document.write(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa);

</script>

</html>

=============================================

Dynamically Typed Language:

===========================

High-level Programming languages:

classified into three types:

1) Statically Typed

2) Strongly Typed

3) Dynamically Typed

1) Statically Typed

===================

==> C, C++ ==> Statically typed programming languages

based the type of the value the assigned value can be automatically modified.

#include<stdio.h>

void main()

{

int a; // variable declaration

a = 123;

a = 1.23f;

printf("%d\n",a);

printf("%f\n",a);

}

2) Strongly Typed

==================

==> we can't access the variable without initialization.

==> we can define the variable with values based on the ranking of the primitive datatypes.

byte << short << int << long << float << double

class Variables{

public static void main(String[] args)

{

int a;

a = 123;

a = 1.23f;

System.out.println(a);

}

}

3) Dynamically Typed

=====================

Ex: JavaScript, Python

==> For dynamically typed languages,

the variable name never be preceded with its type.

Because, the JS can detect its type based on the value which we have assigned.

<!DOCTYPE html>

<html>

<script>

a = 123;

document.write(a);

document.write(typeof(a)); // number

a = 1.23;

document.write(a);

document.write(typeof(a)); // number

a = "hello";

document.write(a);

document.write(typeof(a)); // string

</script>

</html>

=================================================

VARIABLES

=========

==> we have three keywords to define variables:

1) var keyword

2) let keyword

3) const keyword

Note:

====

without these keywords also, we can define variables. But those variable can automatically detect as "var" type.

Hoisting:

=========

is the process to extend the scope of property/variable/function to above the definition.

<!DOCTYPE html>

<html>

<script>

document.write(a);

var a = 100;

document.write(a);

</script>

</html>

Note:

====

We can't observe the hoisting property on the variables which are without any keywords like: var/let/const.

1) var keyword

==============

Syntax:

var identifier = value; // variable declaration and initialization

1) the default value for the variable with var type ==> "undefined".

2) the variable with var type can be hoisted.

3) the variable with var type can be allowed to define within the block and allowed to access within the same block and even in outside block also.

<!DOCTYPE html>

<html>

<script>

document.write(a+"<br>");

var a;

document.write(a+"<br>");

a = 123;

document.write(a+"<br>");

if(true)

{

var b = 121;

document.write(b+"<br>");

}

document.write(b+"<br>");

if(true)

{

document.write(b);

}

</script>

</html>

**Day30: Variable and IO Operations:**

Variables with let keyword:

===========================

Syntax:

let identifier; // variable declaration with let keyword

==> the variable with let is not to be hoisted.

That means, we can't access the variable before the definition.

==> the default value for the variable with let ==> "undefined".

==> the variable with let can able to access within the defined block but we can't extend that variable to outside the same block.

Var Vs let:

===========

1) The variable with var can be hoisted. But the variable with let cannot be hoisted.

2) The variable with var can access within the same block and extend to outside of the block also. Whereas the variable with let we can access within the same block but we can't extend to outside the block.

hoisting:

=========

process to extend the scope the variables/functions to above the definition.

const keyword:

==============

Syntax:

for the variable with const is:

const identifier;

Note:

the constant variable must be declared and initialized within the same line.

==> the constant variable never be hoisted.

==> Constant variable never be considered for the modification.

==> As like the let, the const keyword can able to access within the same block and cannot extend to outside the defined block.

let Vs Const:

=============

1) let can be modified

but const can't be modified.

2) let can be accessed without initialization but const cannot be accessed without initialization.

var Vs Const:

=============

1) var can be hoisted but const cannot be hoisted.

2) var can be accessed without initialization but const cannot be accessed without initialization.

3) var can have block scope and can extend to outside the block also. But const can always have only the block scope.

===============================================

IO Operations:

==============

IO ==> Input and Output

Input ==> reading of the data or accepting of the data or storing of the data

Output ==> writing of the data or printing of the data

In Java:

========

Input ==> Scanner Class

import java.util.Scanner;

main()

{

Scanner obj = new Scanner(System.in);

}

Input Operation:

================

prompt():

=========

==> using this we can read the value for the variable.

==> the prompt() can always read the value in string format.

Syntax:

identifier = prompt();

<!DOCTYPE html>

<html>

<script>

let a = prompt("Enter a value for the variable a:");

document.write("The Value of a = "+a+"<br>");

document.write("The Type of given Variable = "+(typeof a));

</script>

</html>

============================================

Output Operations:

==================

alert()

=======

Syntax:

alert(value);

==> alert() can always printing anything in prompt box.

==> the alert() can always gives the content with confirmed prompt box.

confirm()

=========

Syntax:

confirm(value);

==> confirm() can always print anything in prompt box as same as the alert().

==> confirm() can give optional prompt box with data.

innerTEXT

=========

Syntax:

document.HTMLelement.innerText = "value";

==> can always print anything as web content.

==> But it cannot understand the HTML or CSS styles.

innerHTML

=========

Syntax:

document.HTMLElement.innerHTML = "value";

==> it can always nest the html content within the selected html element.

outerHTML

=========

Syntax:

document.HTMLElement.outerHTML = "value";

==> the selected html element can replace with new html content.

write()

======

Syntax:

document.write("value");

==> inline method

log()

=====

==> can always print anything on the console.

Syntax:

console.log(value);

<!DOCTYPE html>

<html>

<head>

<title> Output Techniques </title>

<script>

function DeleteData()

{

if(confirm("Are you sure want to delete the data?") == true)

{

// alert("The Data which was selected was deleted successfully.");

// document.getElementById("text").innerText = "The Data which was selected was deleted successfully.";

// document.querySelector("h2").innerText = "The Data which was selected was deleted successfully.";

// document.querySelector("#text").innerText = "<p> The Data which was selected was deleted successfully. </p>";

// document.querySelector("#text").innerHTML = "<p> The Data which was selected was deleted successfully. </p>";

document.querySelector("#text").outerHTML = "<p> The Data which was selected was deleted successfully. </p>";

}

else{

alert("The Delete Process was cancelled by Interrupt.");

}

}

</script>

</head>

<body>

<h1> Output Methods </h1>

<button onclick="DeleteData()"> Delete </button>

<h2 id = "text"></h2>

</body>

</html>

**Day31: JavaScript Interview Questions:**

IO OPERATIONS:

==============

prompt() ==> Input Technique

alert(), confirm(), write(), innerText, innerHTML and outerHTML ==> Output Techniques

can prompt() return anything?

=============================

prompt() can return three values:

1) null ==> when we cancel the prompt()

2) "" (empty string) ==> when we can click on "yes" without enter any value

3) value ==> after the value entered

<!DOCTYPE html>

<html>

<head>

<title> Is prompt() return anything? </title>

<script>

function createClick(){

f = prompt("Enter folder name:");

if(f == null){

document.write("Process of folder creation has cancelled.");

}

else if(f == "")

{

document.write("Please Enter valid folder name.");

}

else{

document.querySelector("p").innerHTML += "Folder Created :" + f + "<br>";

}

}

</script>

</head>

<body>

<button onclick="createClick()"> Create Folder </button>

<p></p>

</body>

</html>

============================================

<!DOCTYPE html>

<html>

<script>

s = prompt("Enter a value:");

document.write(s);

</script>

</html>

=======================================================

strict mode of JavaScript:

==========================

==> to avoid the individual coding styles using javascript, we need to enable "strict" mode.

Syntax:

"use strict";

==> allows to reduce the code in-consistency.

and all the developers should follow the standards of the JavaScript.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

"use strict";

x = 10;

document.write(x);

</script>

</body>

</html>

=================================================

How to write Javascript for Legacy Browsers?

============================================

Legacy Browsers ==> Old versioned browsers.

Operator:

Arithmetic Operator ==> \*\* ==> Exponent Operator or Power operator

ex: 2 \*\* 3 ==> 8 (new version)

old version ==> Math.pow(2,3)

==> To instruct about the new javascript to the legacy browsers: we should write the javascript within HTML comments.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=<device-width>, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

<!--

let x = 2

let y = 3

document.write((x \*\* y) +"<br>");

document.write(Math.pow(x,y));

-->

</script>

</body>

</html>

**Day32:** **Datatypes**:

Q: How to differentiate variable in strict mode than the normal mode?

=====================================================================

In general:

variable need not declared in JS when the strict mode is disabled.

==> In strict mode, the variable can be three conditions:

1) Declaration ==> let/var/const

2) Assignment

3) Initialization

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

"use strict";

let a;

a = 100; // assignment

var b = 123; // initialization

document.write("a = "+a+"<br>");

document.write("b = "+b);

</script>

</body>

</html>

==================================================

JavaScript Comments:

====================

==> Comments can be used for readability.

==> Two ways:

1) Single Line Comments ==> //

2) Multi Line Comments ==> /\* comments are here \*/

==> in anywhere of the program.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

// Enable the strict mode

"use strict";

/\*

variable declared with "let type" and with the name "a"

and assigned a value "100".

\*/

let a;

a = 100; // assignment

// Variable declaration and assignment is called as Initialization.

var b = 123; // initialization

document.write("a = "+a+"<br>");

document.write("b = "+b);

// all the variables are printed here

</script>

</body>

</html>

=======================================================

JavaScript Datatypes:

=====================

Datatype ==> type of the data to store in the variable.

ex:

application form:

=================

User name ==> string

age ==> integer

salary ==> double/float etc.

==> Datatypes are classified into two types:

1) Primitive Datatypes

All fundamental datatypes

are always define with single literal

ex: x = 10, salary = 55000.0; etc.

2) Non-Primitive Datatypes

All Collection types

are always define with more than one literal

Ex: "JavaScript", [1,2,3,4,5,6], {'a':10,'b':20}

Primitive Datatypes

===================

Number

Boolean

Number

======

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

"use strict";

var a;

a = 123; // decimal

console.log(typeof a);

// binary ==> base 2 value ==> two literals: 1 and 0

// prefix : 0b/0B

a = 0b110011; // binary

console.log(typeof a);

// octal ==> base-8 number ==> literals: 0 to 7

a = 0O7654321; // octal

console.log(typeof a);

// hexadecimal ==> base16 number ==> literals: 0 to 9 and a to f or A to F

// prefix : 0x / 0X

a = 0X12fa;

console.log(typeof a);

a = 1.234; // floating

console.log(typeof a);

a = 1.2e-5; // scientific data

console.log(typeof a);

</script>

</body>

</html>

=========================================================

Boolean

=======

true, false

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

"use strict";

let a; // variable declaration

a = true;

console.log(typeof a);

a = false;

console.log(typeof a);

console.log(true + false);

console.log(true + true);

</script>

</body>

</html>

===================================================

Non-Primitive Datatypes

========================

Strings

Objects ==> Arrays, Maps

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

/\*

string can define in three ways:

no character data representation

1) using single quotes

2) using double quotes

3) back-tik operator

\*/

"use strict";

let str;

str = 'a'; // string

console.log(typeof str);

str = 'javascript';

console.log(typeof str);

str = "Programming Language";

console.log(typeof str);

str = `JavaScript Compiler`;

console.log(typeof str);

console.log(str[0]);

// str = 1234

// console.log(str[0]);

</script>

</body>

</html>

=============================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

"use strict";

var a;

a = [10,20,30,40,50]; // array

console.log(typeof a);

console.log(a[1]);

a = {'apple':100,'banana':200,'cherry':300}; // map

console.log(typeof a);

console.log(a['banana']);

</script>

</body>

</html>

**Day33: Shadowing, Variable Scope:**

Shadowing:

==========

==> it is a process of re-declaring same name identifier within the scope.

==> there are three different scopes for variables:

1) Block Scope/Local Scope

2) Function Scope

3) Global Scope

4) Module Scope ==> Within OOP

1) Block Scope:

================

==> The variable can able to access within the same block (in which block, we have defined)

"use strict";

var x;

x = 10;

const p = 123;

if(x == 10){

const y = 20;

// y = 20;

console.log("x = "+x+"y = "+y);

console.log("p = "+p);

}

// console.log("x = "+x+"y = "+y);

2) Function Scope

=================

==> The variable which we have defined within the function, and that can allowed to access within the same function. That variable is called as "Function scope variable".

"use strict";

function f1()

{

var x;

let y;

const z = 123;

x = 100;

y = 200;

console.log("x = "+x);

console.log("y = "+y);

console.log("z = "+z);

}

function f2()

{

// console.log("x = "+x);

// console.log("y = "+y);

// console.log("z = "+z);

console.log("Hi");

}

f1();

f2();

======================================

Global Scope:

=============

The variables we can able to access in all the functions of the same programs, those variables are called as "Global Variables".

"use strict";

var x = 100;

let y = 200;

const z = 300;

function f1()

{

console.log("x = "+x);

console.log("y = "+y);

console.log("z = "+z);

}

function f2()

{

console.log("x = "+x);

console.log("y = "+y);

console.log("z = "+z);

}

f1();

f2();

Q: Do we able to change the global variable?

=============================================

yes.

"use strict";

var x = 100;

let y = 200;

const z = 300;

function f1()

{

console.log("x = "+x);

console.log("y = "+y);

console.log("z = "+z);

}

function f2()

{

var x = 101;

y = 202;

// z = 303;

console.log("x = "+x);

console.log("y = "+y);

console.log("z = "+z);

}

function f3()

{

console.log("x = "+x);

console.log("y = "+y);

console.log("z = "+z);

}

f1();

f2();

f3();

f1();

===============================

Note:

====

Shadowing is applicable/allowed for only var but not to let and const.

"use strict";

var x = 10;

let y = 20;

const z = 30; // re-assignment/assignment is not possible

console.log(x);

console.log(y);

console.log(z);

// Assignments

x = 20;

y = 30;

// z = 40;

console.log(x);

console.log(y);

console.log(z);

// Shadowing ==> Re-declaration

var x; // re-declared

// let y;

console.log(x);

console.log(y);

console.log(z);

===============================================

Q: CAN WE DECLARE A GLOBAL VARIABLE INSIDE THE FUNCTION?

========================================================

yes.

"window" object ==> browser

"use strict";

function f1(){

window.x = 123;

window.y = 321;

document.write("x = "+x);

document.write("y = "+y);

}

function f2(){

document.write("x = "+x);

document.write("y = "+y);

}

f1();

f2();

======================================================

Q: Do we modify the constant?

=============================

Yes, only dynamically.

"use strict";

// const z = 123;

// console.log(z)

// z = 321;

// console.log(z)

const name = prompt("Enter your name:")

console.log("Hi"+name);

**Day34: Type Casting:**

TYPE CASTING:

=============

==> ALSO CALLED AS "TYPE CONVERSION"

==> CONVERTING THE PRIMITIVE DATA TO ANOTHER PRIMITIVE TYPE OF THE DATA.

==> TWO TYPES:

1) IMPLICIT TYPE CASTING/AUTOMATIC TYPE CASTING

2) EXPLICIT TYPE CASTING

1) IMPLICIT TYPE CASTING/AUTOMATIC TYPE CASTING

================================================

Based on the type of inputs and operator, the JS compiler can automatically perform conversion is called as "Automatic Type Casting".

console.log(typeof ("JavaScript" + 123));

let a = 'JavaScript';

var b = 123;

let c = "Java"+b

console.log(a+b);

console.log(c);

2) EXPLICIT TYPE CASTING

=========================

Number()

=======

var a = "123";

var b = "123.123";

var c = "abcd";

var d = 123.234;

var e = true;

// Number()

console.log(typeof a);

console.log(typeof b);

console.log(typeof c);

console.log(typeof Number(a));

console.log(typeof Number(b));

console.log(typeof Number(c));

console.log(Number(d));

console.log(Number(e));

===========================================

parsetInt()

===========

var a = "123";

var b = "123.123";

var c = "abcd";

var d = 123.234;

var e = true;

// parseInt()

console.log(typeof parseInt(a));

console.log(typeof parseInt(b));

console.log(parseInt(c));

console.log(typeof parseInt(d));

console.log(typeof parseInt(e));

parseFloat()

String()

Boolean()

console.log(typeof String(123));

console.log(typeof String(true));

console.log(Boolean(-123))

console.log(Boolean(123.123))

console.log(Boolean(0.0000))

console.log(typeof Boolean(''))

console.log(Boolean('123'))

========================================================

Why back tick for strings?

===========================

<!DOCTYPE html>

<html>

<script>

let age = parseInt(prompt("Enter your age:"));

if(isNaN(age))

{

document.write("Enter the value in nunber format");

}

else

{

//document.write("Your Age is "+(parseInt(age)+1)+" by next year");

document.write(`Your Age is ${parseInt(age)+1} by next Year`);

}

</script>

</html>

**Day35: String Formatting Methods:**

String Handling

===============

CSS

color text ==> color

font size

1) String Formatting methods:

=============================

==> To format the strings, JS can provide built-in methods:

1) bold()

=========

==> define the boldness to the text/string, we can use the "bold()".

Syntax:

string-object/string-variable/string-data.bold()

2) italics()

============

==> to apply italic nature to the string/text, we can use "italics()".

Syntax:

string-object.italics()

3) fontcolor()

==============

==> to define the color to the string, we can use "fontcolor()".

Syntax:

string-object.fontcolor('color-value')

4) fontsize()

==============

==> used to increase or decrease the font of the given string/text.

Syntax:

string-object.fontsize('value\_integer')

5) toLowerCase()

================

==> to change the case of the string to lower case, we can use "toLowerCase()".

Syntax:

string-object.toLowerCase()

6) toUpperCase():

=================

==> used to change the case of the string to upper case.

Syntax:

string-object.toUpperCase()

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function clickSubmit()

{

var username = document.getElementById("user").value;

document.write(typeof username+"<br>");

document.write(username+"<br>");

// username.bold();

var username1 = username.bold().italics().fontcolor('blue').fontsize('20').toUpperCase();

document.write(username1+"<br>");

}

</script>

</head>

<body>

<h1> User Registration </h1>

<dl>

<dt> User Name </dt>

<dd>

<input type = "text" id = "user">

</dd>

</dl>

<button onclick="clickSubmit()"> Submit </button>

</body>

</html>

Reason:

========

to define the string formatting's in dynamically or in run-time, we can use "string formatting methods" from the JavaScript.

==========================================================

Property:

=========

length:

=======

we can use to find the number of characters of the given string.

Syntax:

string-object.length

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function clickSubmit()

{

var username = document.getElementById("user").value;

var msg = document.getElementById("message");

if(username.length >= 4 && username.length <= 10)

{

msg.innerHTML = "User name is valid.".bold().fontcolor('blue');

}

else

{

msg.innerHTML = "User name is invalid.".bold().fontcolor('red');

}

}

</script>

</head>

<body>

<h1> User Registration </h1>

<dl>

<dt> User Name </dt>

<dd>

<input type = "text" placeholder="Block Letters Only" id = "user" onkeyup="clickSubmit()">

</dd>

<dd id = "message"></dd>

</dl>

<button onclick="clickSubmit()"> Submit </button>

</body>

</html>

===================================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function clickSubmit()

{

var username = document.getElementById("user").value;

var msg = document.getElementById("message");

if(username.length >= 4 && username.length <= 10)

{

msg.innerHTML = "User name is valid.".bold().fontcolor('blue');

}

else

{

msg.innerHTML = "User name is invalid.".bold().fontcolor('red');

}

}

function caseChange()

{

var username = document.getElementById("user").value;

document.getElementById("user").value = username.toUpperCase();

}

</script>

</head>

<body>

<h1> User Registration </h1>

<dl>

<dt> User Name </dt>

<dd>

<input type = "text" placeholder="Block Letters Only" id = "user" onblur="caseChange()">

</dd>

<dd id = "message"></dd>

</dl>

<button onclick="clickSubmit()"> Submit </button>

</body>

</html>

**Day36: Practice On Strings:**

Events:

=======

1) onclick:

===========

==> can allowed to perform some action/actions after the click

2) onload:

==========

==> can allowed to perform some action/actions after the loading of the page or image.

3) onkeyup:

===========

==> help to perform the actions while releasing the key

4) onblur:

==========

==> actions can be performed when the element in blur state.

Changing/applying the color to the text in run-time/dynamically:

=================================================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function changeColor(){

// document.querySelector("h1").fontcolor(document.querySelector("select").value);

document.querySelector("h1").style.color = document.querySelector("select").value;

}

</script>

</head>

<body>

<fieldset>

<legend> Chose Effects </legend>

<dl>

<dt> Font Color </dt>

<dd>

<select onchange="changeColor()">

<option> Red </option>

<option> Blue </option>

<option> Green </option>

<option> White </option>

<option> Yellow </option>

</select>

</dd>

</dl>

</fieldset>

<h1> Welcome To Ashok IT </h1>

</body>

</html>

=========================================================

Dark theme to light theme:

==========================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

.form{

border: 2px solid black;

padding: 20px;

width: 250px;

margin-top: 20px;

}

.dark-theme{

background-color: black;

color: white;

}

.light-theme{

background-color: white;

color: black;

}

</style>

<script>

function changeTheme(){

var themeCheckBox = document.getElementById("theme");

var formContainer = document.getElementById("formContainer");

if(themeCheckBox.checked)

{

formContainer.className = "dark-theme";

}

else{

formContainer.className = "light-theme";

}

}

</script>

</head>

<body class = "container-fluid">

<div class = "form-switch">

<input type = "checkbox" id = "theme" onchange = "changeTheme()" class = "form-check-input">

Dark Theme

</div>

<div class = "form" id = "formContainer">

<h1> User Registration Form </h1>

<dl>

<dt> User Name </dt>

<dd>

<input type = "text">

</dd>

<dt> Password </dt>

<dd>

<input type="password">

</dd>

<dt> Email </dt>

<dd>

<input type="email">

</dd>

</dl>

<button> Register </button>

</div>

</body>

</html>

**Day37: String Handling Methods Part\_01:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

.form{

border: 2px solid black;

padding: 20px;

width: 250px;

margin-top: 20px;

}

.dark-theme{

border: 2px solid black;

padding: 20px;

width: 250px;

margin-top: 20px;

background-color: black;

color: white;

}

.light-theme{

border: 2px solid black;

padding: 20px;

width: 250px;

margin-top: 20px;

background-color: white;

color: black;

}

</style>

<script>

function changeTheme(){

var themeCheckBox = document.getElementById("theme");

var formContainer = document.getElementById("formContainer");

if(themeCheckBox.checked)

{

formContainer.className = "dark-theme";

}

else{

formContainer.className = "light-theme";

}

}

</script>

</head>

<body class = "container-fluid">

<div class = "form-switch">

<input type = "checkbox" id = "theme" onchange = "changeTheme()" class = "form-check-input">

Dark Theme

</div>

<div class = "form" id = "formContainer">

<h1> User Registration Form </h1>

<dl>

<dt> User Name </dt>

<dd>

<input type = "text">

</dd>

<dt> Password </dt>

<dd>

<input type="password">

</dd>

<dt> Email </dt>

<dd>

<input type="email">

</dd>

</dl>

<button> Register </button>

</div>

</body>

</html>

String Manipulations:

=====================

1) charAt():

============

==> a built-in function in javascript,

which accepts an index

and return the character at that index.

Syntax:

string-object.charAt(index);

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var name = "Ashok IT";

document.write(name.charAt(0)+"<br>");

document.write(name.charAt(1)+"<br>");

document.write(name.charAt(2)+"<br>");

document.write(name.charAt(3)+"<br>");

document.write(name.charAt(4)+"<br>");

document.write(name.charAt(5)+"<br>");

document.write(name.charAt(6)+"<br>");

document.write(name.charAt(7)+"<br>");

document.write("<br>");

document.write(name.charAt(-1)+"<br>");

</script>

</body>

</html>

==============================

2) charCodeAt():

================

256 characters ==> 0 to 255

ASCII ==> American Standard Code Information Interchange ==> 1-byte

Java/Python/JavaScript ==> does support more than 256 characters

Ex: java:

char ==> 2-bytes ==> 16-bits ==> 65536 chars ==> Unicodes

0 to 65535

char ch = 'a';

==> charCodeAt() ==> a built-in function

can be used to accept an index

and return the Unicode value of the character which presents at that specified index.

Syntax:

str-obj.charCodeAt(index)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var name = "Ashok IT";

document.write(name.charCodeAt(0)+"<br>");

document.write(name.charCodeAt(1)+"<br>");

document.write(name.charCodeAt(2)+"<br>");

document.write(name.charCodeAt(3)+"<br>");

document.write(name.charCodeAt(4)+"<br>");

document.write(name.charCodeAt(5)+"<br>");

document.write(name.charCodeAt(6)+"<br>");

document.write(name.charCodeAt(7)+"<br>");

document.write("<br>");

document.write(name.charAt(-1)+"<br>");

</script>

</body>

</html>

======================================

Acquiring the part of the string:

=================================

"JavaScript"

"Java"

"Script"

slice():

========

Syntax:

str-obj.slice(start-index)

==========================

==> can return group of characters from the specified index (start) to till last character.

str-obj.slice(start-index, end-index)

=====================================

==> can return group of characters from the specified index range.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var str = "JavaScript";

var part1 = str.slice(1);

var part2 = str.slice(0,4);

var part3 = str.slice(4);

var part4 = str.slice(4,0);

var part5 = str.slice(-5,-1);

document.write("Original String = "+str+"<br>");

document.write("Part of the string = "+part1+"<br>");

document.write("Part of the string = "+part2+"<br>");

document.write("Part of the string = "+part3+"<br>");

document.write("Part of the string = "+part4+"<br>");

document.write("Part of the string = "+part5+"<br>");

</script>

</body>

</html>

==> slice() function can always acquire the part of the string in only forward direction (left to right)

Reverse Direction ==> right to left ==> negative index values ==> -1 to -total-number-chars

Forward Direction ==> left to right ==> positive index values ==> 0 to tota\_chars - 1

substr():

========

Syntax:

str.substr(start-index)

substr(start-index, end-index)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var str = "JavaScript";

var part1 = str.substr(3);

var part2 = str.substr(0,4);

var part3 = str.substr(4,0); // not possible with reverse accessing

var part4 = str.substr(-5,-1); // not understand the negative index values

var part5 = str.substr(-1,-5);

document.write("Original String = "+str+"<br>");

document.write("Part of the string = "+part1+"<br>");

document.write("Part of the string = "+part2+"<br>");

document.write("Part of the string = "+part3+"<br>");

document.write("Part of the string = "+part4+"<br>");

document.write("Part of the string = "+part5+"<br>");

</script>

</body>

</html>

substring():

============

Syntax:

str.substring(start-index)

str.substring(start-index, end-index)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var str = "JavaScript";

var part1 = str.substring(0);

var part2 = str.substring(4,10);

var part3 = str.substring(-5,-1); // negative index not supported

var part4 = str.substring(10,4); // can do reverse acquiring

document.write("Original String = "+str+"<br>");

document.write("Part of the string = "+part1+"<br>");

document.write("Part of the string = "+part2+"<br>");

document.write("Part of the string = "+part3+"<br>");

document.write("Part of the string = "+part4+"<br>");

</script>

</body>

</html>

**Day38: String Handling Part\_03:**

String Manipulation Functions:

==============================

1) charAt()

2) charCodeAt()

3) slice()

4) substr()

5) substring()

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function changeCase(){

// sentence case ==> Capitalize case

// the first letter of the sentence ==> Upper case and remaining all till the last char

// are in lower case

let sentence = document.getElementById("user").value;

let firstCharacter = sentence.charAt(0);

let remCharacters = sentence.substring(1);

let part1 = firstCharacter.toLocaleUpperCase();

let part2 = remCharacters.toLowerCase();

let sentenceCase = part1 + part2;

document.getElementById("user").value = sentenceCase;

}

</script>

</head>

<body>

<fieldset>

<legend> String Conversion </legend>

<div>

<input type="text" id = "user" size = "40" onblur="changeCase()">

</div>

</fieldset>

</body>

</html>

===========================================

indexOf()

=========

==> indexOf() can accept a character and return the index of the specified character.

Syntax:

str-object.indexOf(character)

==> if the specified character is duplicated (char at multiple places within the string):

indexOf() can return the index of the first occurrence.

==> if the specified character is not present:

indexOf() can return '-1'

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

str = "welcome to javascript";

let index = str.indexOf('t');

let duplicate = str.indexOf('o');

let absent = str.indexOf('R');

document.write(index+"<br>");

document.write(duplicate+"<br>");

document.write(absent);

</script>

</body>

</html>

=====================================

lastIndexOf():

==============

==> lastIndexOf() can accept a character and return the index of the specified character.

Syntax:

str.lastIndexOf(character)

==> if the specified character is duplicated:

lastIndexOf() can return:

character's the last occurrence index

==> if the specified character is not present:

it can return -1.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

str = "welcome to javascript";

let index = str.lastIndexOf('i');

let duplicate = str.lastIndexOf('o');

let absent = str.lastIndexOf('R');

document.write(index+"<br>");

document.write(duplicate+"<br>");

document.write(absent);

</script>

</body>

</html>

===================================================

<!-- WAP using javascript to check whether the given email id is valid or not. -->

<!-- Hint:

if the @ is after 3 character in the string ==> valid.-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function emailVerification(){

var mailId = document.getElementById("mail").value;

let index = mailId.indexOf('@');

if(index > 2){

document.querySelector("h1").innerHTML = "Email Verified Successfully";

}

else{

document.querySelector("h1").innerHTML = "Email is Invalid";

}

}

</script>

</head>

<body>

<fieldset>

<legend> Email </legend>

<div>

<input type="email" id = "mail" size = "40">

</div><br>

<button onclick="emailVerification()"> SUbmit </button>

</fieldset>

<h1></h1>

</body>

</html>

**Day39: String Handling Part\_05:**

String Manipulation Functions:

==============================

1) charAt()

2) charCodeAt()

3) slice()

4) substr()

5) substring()

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function changeCase(){

// sentence case ==> Capitalize case

// the first letter of the sentence ==> Upper case and remaining all till the last char

// are in lower case

let sentence = document.getElementById("user").value;

let firstCharacter = sentence.charAt(0);

let remCharacters = sentence.substring(1);

let part1 = firstCharacter.toLocaleUpperCase();

let part2 = remCharacters.toLowerCase();

let sentenceCase = part1 + part2;

document.getElementById("user").value = sentenceCase;

}

</script>

</head>

<body>

<fieldset>

<legend> String Conversion </legend>

<div>

<input type="text" id = "user" size = "40" onblur="changeCase()">

</div>

</fieldset>

</body>

</html>

===========================================

indexOf()

=========

==> indexOf() can accept a character and return the index of the specified character.

Syntax:

str-object.indexOf(character)

==> if the specified character is duplicated (char at multiple places within the string):

indexOf() can return the index of the first occurrence.

==> if the specified character is not present:

indexOf() can return '-1'

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

str = "welcome to javascript";

let index = str.indexOf('t');

let duplicate = str.indexOf('o');

let absent = str.indexOf('R');

document.write(index+"<br>");

document.write(duplicate+"<br>");

document.write(absent);

</script>

</body>

</html>

=====================================

lastIndexOf():

==============

==> lastIndexOf() can accept a character and return the index of the specified character.

Syntax:

str.lastIndexOf(character)

==> if the specified character is duplicated:

lastIndexOf() can return:

character's the last occurrence index

==> if the specified character is not present:

it can return -1.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

str = "welcome to javascript";

let index = str.lastIndexOf('i');

let duplicate = str.lastIndexOf('o');

let absent = str.lastIndexOf('R');

document.write(index+"<br>");

document.write(duplicate+"<br>");

document.write(absent);

</script>

</body>

</html>

===================================================

<!-- WAP using javascript to check whether the given email id is valid or not. -->

<!-- Hint:

if the @ is after 3 character in the string ==> valid.-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function emailVerification(){

var mailId = document.getElementById("mail").value;

let index = mailId.indexOf('@');

if(index > 2){

document.querySelector("h1").innerHTML = "Email Verified Successfully";

}

else{

document.querySelector("h1").innerHTML = "Email is Invalid";

}

}

</script>

</head>

<body>

<fieldset>

<legend> Email </legend>

<div>

<input type="email" id = "mail" size = "40">

</div><br>

<button onclick="emailVerification()"> SUbmit </button>

</fieldset>

<h1></h1>

</body>

</html>

**Day40**: **Primitive Datatypes Remaining**:

Password Validation with meter tag:

===================================

<meter> tag:

============

==> scale indication tag

==> never display any text

==> pair tag

must required with open and close tags

==> attributes allowed for <meter> tag are:

1) id

2) class

3) min ==> minimum value for scale indication

4) max ==> maximum value for scale indication

Syntax:

<meter id = "value" class = "value" min = "numeral" max = "numeral"></meter>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function showGrade(min, max, value){

grade.min = min;

grade.max = max;

grade.value = value;

}

function passwordValidation(){

var grade = document.getElementById("grade");

var password = document.getElementById("textPassword").value;

var message = document.getElementById("passwordMessage");

var regExp = /(?=.\*[A-Z]\w{0,9})/;

if(password.match(regExp)){

message.innerHTML = "Strong Password";

showGrade(1,100,100);

}

else{

if(password.length < 4){

message.innerHTML = "Poor Password";

showGrade(1,100,20);

}

else{

message.innerHTML = "Weak Password";

showGrade(1,100,70);

}

}

}

</script>

</head>

<body>

<h2> Password Verification </h2>

<input type = "password" id = "textPassword" onkeyup="passwordValidation()">

<div id = "passwordMessage"></div>

<div>

<meter id = "grade" min = "1" max = "100"></meter>

</div>

</body>

</html>

======================================================

Boolean Type:

=============

==> is one of the primitive datatype in javascript.

==> can be define with two values:

1) true

2) false

==> can use in decision makings.

==> JS compiler can understand

true ==> 1

false ==> 0

true + true ==> 2

true + false ==> 1

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var brand = "Samsung Galaxy S60";

var stock = true;

var sales;

document.write(`Brand = ${brand} <br> stock = ${stock}<br>`);

if(stock == true){

sales = "Available for sales";

}

else{

sales = "Out of stock";

}

document.write(sales);

</script>

</body>

</html>

=============================================

Undefined Type:

===============

reference ==> there

value assignment ==> not there

undefined ==> Keyword

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var product = "Samsung TV"; // value assignment

var price; // no value assignment

document.write(typeof product);

document.write("<br>");

document.write(typeof price);

document.write("<br>");

// document.write(`My Product = ${product} and price of the product = ${price}<br>`);

if(price == undefined){

document.write(`My Product = ${product}`);

}

else{

document.write(`My Product = ${product} and price of the product = ${price}<br>`);

}

</script>

</body>

</html>

=========================================

Q: What is the difference between undefined and not-defined?

============================================================

undefined ==> variable reference is available but no value assignment

not-defined ==> variable reference is not available. ==> Kind of Exception (Error)

var x = 100;

var y;

console.log("x = ",x);

console.log("y = ",y);

console.log("z = ",z);

==============================================

Null Type:

==========

null ==> keyword

==> null statement:

variable reference ==> available

value assignment ==> not available.

==> null can be detected in run time.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var userName = prompt("Enter your name:");

// document.write(`User name = ${userName}`);

if(userName == ""){

document.write("Please Enter Your Name:");

}

else if(userName == null){

document.write("You got cancelled");

}

else{

document.write("Your Name = "+userName);

}

</script>

</body>

</html>

**Day41: Array Handling Part\_01:**

Numbers ==> Numeral values

Strings ==> Literals

Boolean ==> true/false

Undefined

null

Immutable Types Vs Mutable Types:

=================================

Primitive Types ==> Immutable Types

Immutable ==> data once we can define,

we cannot be allow for modification

string = "JavaScript";

console.log(string[0]);

console.log(string);

string[0] = 'j';;

console.log(string);

Non-Primitive Datatypes:

========================

==> Collections

==> Mutable

after the definition we can allow for modification.

==> Non-primitive datatypes are:

1) Array

2) Object

3) Map

Array Type

===========

How we can understand an array?

===============================

form a queue form a stack form an array

==============================================================

Aakash Browser 3-people were

Ashwin History Ashwin ==> 9am ==> 11 am

Asha Asha ==> 9.30am ==> 10am

Jyothi Aarohi ==> 8.30 am ==> 12 noon

FIFO ==> First Input First Output

LIFO ==> Last Input First Output

==> array is collection of any type of elements

array elements should be order

but accessing should be random.

Why we need arrays?

===================

==> for programming perspective,

arrays can be used for reducing the overhead and complexity.

Electronic Devices

Storage

Magnetic Tapes Disk/DVD/Floppy

Java ==> Strongly Typed

C/C++ ==> Statically Typed

JS/Python ==> Dynamically typed

==========================================

Is array size dynamically changed?

==================================

Yes.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let a = new Array(2);

a[0] = 11;

a[1] = 22;

document.write(a+"<br>");

a[2] = 33;

a[3] = 44;

a[4] = 55;

document.write(a+"<br>");

</script>

</body>

</html>

=================================================

How to configure arrays:

========================

1) Array declaration

2) Array initialization

using []:

========

Syntax:

let/var/const arrayName;

arrayName = []; // js compiler, assigning the memory to the array

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let array; // variable declaration

array = []; // memory assignment

array[0] = 10;

array[1] = 20;

array[2] = 30;

document.write(array);

</script>

</body>

</html>

========================================================

using Array():

=============

Syntax:

let/var/const arrayName = new Array();

or

let/var/const arrayName = new Array(size);

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let a1 = new Array();

// document.write(a1);

a1[0] = 11;

a1[1] = 22;

a1[2] = 33;

document.write(a1+"<br>");

var a2 = new Array(5);

a2[0] = 100;

a2[1] = 200;

a2[2] = 300;

a2[3] = 400;

a2[4] = 500;

document.write(a2);

</script>

</body>

</html>

=======================================

Array Initialization:

=====================

let a; // declaration

a = 10; // assignment

var b = 20; // initialization

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

var products = ["TV","Mobile","Shoes","Washing-Machine"];

document.write(products)

</script>

</body>

</html>

========================================

Index Vs Property:

=================

Arrays in JS ==> not been specified with index

those can be specified with property

property ==> string type

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let products = [100,"Mobiles","Shoes"];

for(i in products){

document.write(typeof i +"<br>");

}

</script>

</body>

</html>

==========================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let product = [];

product[0] = "TV";

product["1"] = "Mobile";

document.write(product);

</script>

</body>

</html>

**Day42: Arrays Part\_02:**

Reading of elements of an array:

================================

using methods

=============

1) toString():

==============

==> a built-in function in JS

can be used to convert an array into string.

Syntax:

array-name.toString()

let sales = [56000, 39995, 43398, 19796, 97890, 59987];

console.log("The Array with Sales (Original Array) = ",sales);

// convert the array into string

sales\_result = sales.toString();

console.log("The Array in String Format = ",sales\_result);

console.log("The Type = ",typeof sales\_result);

console.log("The Type of definition = ",typeof sales);

2) join():

==========

==> when the given array need to convert into the string by joining all the individual elements of an array with the specified delimiter (separator), we can use "join()".

Syntax:

array-name.join('separator')

let sales = [56000, 39995, 43398, 19796, 97890, 59987];

console.log("The Array with Sales (Original Array) = ",sales);

// joining of array elements into string with specified separator.

sales\_join = sales.join('-->');

console.log("The Given Array with separator = ",sales\_join);

console.log("The Type of Result = ",typeof sales\_join);

3) slice():

===========

==> slice() is a built-in function

which can be used to acquire the part of the array (from the specified index range).

Syntax:

array-name.slice(start-index, end-index)

Here:

from end-index: while the slicing

we can reduce the one

let sales = [56000, 39995, 43398, 19796, 97890, 59987];

console.log("The Array with Sales (Original Array) = ",sales);

res1 = sales.slice(2); // from index 2 till the end all elements can acquire.

res2 = sales.slice(1,5); // acquire the array elements with index 1, 2, 3, 4

console.log("Result1 = ",res1);

console.log("Result2 = ",res2);

console.log("The Type of result1 = ",typeof res1);

console.log("The Type of result2 = ",typeof res2);

4) filter():

============

==> can create new array object with elements which are based on the specified condition.

==> the condition for filter() must be define with another function.

Syntax:

array-name.filter(function(){// logic});

let sales = [56000, 39995, 43398, 19796, 97890, 59987];

console.log("The Array with Sales (Original Array) = ",sales);

// filter condition

function getData(value){

return value > 50000;

}

filtered\_array = sales.filter(getData);

console.log("The Array after the filter = ",filtered\_array);

resultant\_array = sales.filter(function(sale){return sale < 50000});

console.log("The Resultant Array = ",resultant\_array);

5) find():

=========

==> based on the condition it can return the first occurred element from the given array.

Syntax:

array-name.find(function(){//logic});

let sales = ['56000', '39995', 43398, 19796, 97890, 59987];

console.log("The Array with Sales (Original Array) = ",sales);

// filtered condition

function getData(value){

return value > 50000;

}

let res1 = sales.find(getData);

console.log(res1);

console.log(typeof res1);

let res2 = sales.find(function(data){return data < 50000});

console.log(res2);

console.log(typeof res2);

6) map()

========

Syntax:

array-name.map(function(){//logic});

let sales = ['56000', '39995', 43398, 19796, 97890, 59987];

console.log("The Array with Sales (Original Array) = ",sales);

console.log("All array elements = ");

sales.map(function(value){

document.write(`<li> ${value} </li>`);

})

=====================================================

using loops

===========

for loop

while loop

do while loop

for loop

========

Syntax:

for(var iteration-variable = 0;iteration-variable < array.length;++iterator-variable){

//logic

}

let sales = ['56000', '39995', 43398, 19796, 97890, 59987];

console.log("The Array Elements = ");

// forward accessing

for(var sale = 0;sale < sales.length;++sale)

{

console.log(sales[sale]);

}

console.log();

// reverse Accessing

for(var s = sales.length - 1;s >= 0;--s){

console.log(sales[s]);

}

while loop:

===========

Syntax:

initialization = 0

while(initialization < array.lenth){

//logic

// update

}

let sales = ['56000', '39995', 43398, 19796, 97890, 59987];

console.log("The Array Elements = ");

// forward accessing

let i = 0;

while(i < sales.length){

console.log(sales[i]);

i++;

}

console.log();

// reverse Accessing

j = sales.length - 1;

while(j >= 0){

console.log(sales[j]);

j--;

}

===================================================

using iterators

===============

two different iterators

1) for-in

=========

an iterator used to access/read only the properties of array.

==> to get values using for-in, we should use the []

let sales = ['56000', '39995', 43398, 19796, 97890, 59987];

console.log("The Array Elements = ");

for(var s in sales){

console.log(`${s} --> ${sales[s]}`);

}

2) for-of

==========

is an iterator can use to access the array element values.

let sales = ['56000', '39995', 43398, 19796, 97890, 59987];

console.log("The Array Elements = ");

for(var s of sales){

console.log(s);

}

**Day43: Arrays Part\_03:**

How to create HTML elements using JavaScript Dynamically:

=========================================================

1) Create an HTML Element using createElement():

===============================================

Syntax:

document.createElement("HTML-Element-Name");

2) Add created HTML Element to the web document:

================================================

appendChild()

=============

==> when we want to add the created HTML element into another HTML element, we can use "appendChile()".

ex:

<div> ==> parent tag

<img> ==> child tag

</div>

Syntax:

element.appendChild(Element-name\_reference);

append()

=========

Syntax:

element.append(Element-name\_reference);

prepend()

=========

Syntax:

element.prepend(Element-name\_reference);

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function addImage(){

// creating an Image element using createElement() function from DOM.

var image = document.createElement("img");

image.src = "product1.jpg";

image.width = "400";

image.height = "400";

image.alt = "Image Not Loading";

document.getElementById("container").prepend(image);

}

</script>

</head>

<body>

<div>

<button onclick="addImage()"> Click Here to Add Image </button>

</div>

<div id = "container"></div>

</body>

</html>

========================================================

2) How to add array of elements into web document:

==================================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function addImages(){

var products = ["product1.jpg","product2.jpg","product3.jpg","product4.jpg"];

for(var path of products){

var pic = document.createElement("img");

pic.src = path;

pic.width = "200";

pic.height = "200";

document.getElementById("container").append(pic);

}

}

</script>

</head>

<body onload = "addImages()">

<div id = "container"></div>

</body>

</html>

==========================================================

How to add dropdown dynamically to webpage:

===========================================

<select>

<option> value </option>

</select>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

function dropDown(){

var products = ["product1.jpg","product2.jpg","product3.jpg","product4.jpg"];

for(var path of products){

var option = document.createElement("option");

option.text = path;

document.querySelector("select").append(option);

}

}

</script>

</head>

<body onload="dropDown()">

<h2> Select Path </h2>

<select></select>

</body>

</html>

**Day44: Arrays Part\_04:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

var categories = ["Electronics", "Footwear", "Fashions"];

function bodyLoad(){

for(var category of categories){

var li = document.createElement("li");

li.innerHTML = category;

document.querySelector("ol").appendChild(li);

var option = document.createElement("option");

option.text = category;

document.querySelector("select").appendChild(option);

}

}

</script>

</head>

<body onload = "bodyLoad()">

<div class = "container-fluid">

<div class = "row">

<div class = "col">

<h3> Category List </h3>

<ol></ol>

</div>

<div class = "col">

<h3> Select Category </h3>

<select></select>

</div>

</div>

</div>

</body>

</html>

===================================================================

Q: WHAT TYPE OF DATA CAN STORE IN AN ARRAY?

============================================

ARRAYS CAN ALLOW TO DEFINE WITH ANY TYPE OF THE DATA LIKE:

WITH PRIMITIVE TYPES

WITH NON-PRIMITIVE TYPES

WITH FUNCTIONS.

let array1 = [10,20,30,40,50]; // array with numbers

let array2 = [[1,3,5,7],[2,4,6,8]]; // array with arrays

let array3 = [100,true,'A',[100,200,300]];

let array4 = [function(){console.log("Hi")},function(){console.log("Good Morning")},

function(){console.log("Welcome To Ashok IT")}];

console.log(array1[0])

console.log(array2[1])

console.log(array2[1][2])

console.log(array3[3])

console.log(array4[0]);

======================================================

Q: Why we need to define arrays with functions?

===============================================

==> to handle callback mechanism

==> callback mechanism is a technique where function will execute according to the situation.

["admin",function(){document.write("Login Success")},function(){document.write("Invalid data")}];

Array destruction:

==================

var values = [10,20,30,40,50];

// accessing array elements without array destruction

console.log(values[0]);

console.log(values[1]);

console.log(values[2]);

console.log(values[3]);

console.log(values[4]);

// accessing array elements with array destrcution

var [v1,v2,v3,v4,v5] = values

console.log(v1);

console.log(v2);

console.log(v3);

console.log(v4);

console.log(v5);

==========================

var authorize = ["admin123",function(){console.log("Login Success")},function(){console.log("Login Fail")}];

// array destrcution

var [password,success,failure] = authorize;

var Yourpassword = "admin123";

if(Yourpassword == password){

success();

}

else{

failure();

}

===========================================

Anonymous Function:

===================

==> Name less function

function sumTwoNumbers(a,b){

var result = a+b;

console.log("The Sum = ",result);

}

const message = function(){

console.log("Hi");

}

sumTwoNumbers(100,200);

message();

x = sumTwoNumbers

x(2000,3000);

=========================================

How to add elements into an array:

==================================

push():

======

==> can able to add an element into an array at the end.

Syntax:

array\_name.push(element);

var a = [];

console.log("The Array a = ",a);

a.push(100);

a.push(200);

a.push(300);

console.log("The Array a = ",a);

// do we add more than one element?

a.push(400,500,600,700);

console.log("The Array a = ",a);

============================================================

unshift():

==========

==> can be used to add elements into an array at the beginning.

Syntax:

ArrayName.unshift(element);

var a = [];

console.log("The Array before the unshift operation = ");

console.log(a);

a.unshift(1.2);

a.unshift(2.4);

a.unshift(4.9);

console.log("The Array after the unshift operation = ");

console.log(a);

a.unshift(100,200,300);

console.log("The Array after the unshift operation = ");

console.log(a);

=================================================

splice():

=========

==> can use to add an element at the specified index of an array.

Syntax:

array-name.splice(index, count, element)

here:

index ==> at which index the element want to add

count ==> the number of elements at the specified index want to remove

element ==> element value

var a = [100,111,200,222,300,333];

console.log("Before the Splice Operation:")

console.log("The Array = ",a);

a.splice(2,0,"Mango");

console.log("After the Splice Operation:")

console.log("The Array = ",a);

a.splice(3,3,"Apple");

console.log("After the Splice Operation:")

console.log("The Array = ",a);

a.splice(2,2,"Hi","Hello","Good Morning","Good Afternoon");

console.log("After the Splice Operation:")

console.log("The Array = ",a);

**Day45**: **Arrays Part\_05**:

How to Remove elements from an array:

=====================================

1) pop():

=========

==> can use to remove/return the last element of an array

Ex: [10,20,30,40]

pop() ==> 40

Syntax:

array-name-object.pop()

let a = [100,true,'A','R','B','JavaScript',2000,2024]

console.log("The Given array = ");

console.log(a);

var return1 = a.pop()

console.log("The Array after the pop operation is = ");

console.log(a);

console.log("The Returned value = ",return1);

console.log("The Returned Value = ",a.pop());

console.log("The Array after the pop operation is = ");

console.log(a);

How to understand the return value of the given pre-defined function?

======================================================================

two ways:

1) use the function call to the variable

and print that variable.

2) directly define the function call in print methods like: log(), write() etc.

Note:

=====

using pop(), we cannot remove the specified element and also not possible to remove more than one element.

let a = [100,true,'A','R','B','JavaScript',2000,2024]

console.log("The Given array = ");

console.log(a);

a.pop(true,100,'A');

console.log("The Array after the delete operation is = ");

console.log(a);

====================================================================

2) shift():

===========

==> can use to remove/return the first element of an array.

Syntax:

array-name.shift();

let a = [100,true,'A','R','B','JavaScript',2000,2024]

console.log("The Given array = ");

console.log(a);

// shift operation

console.log("The Deleted Element = ",a.shift()); // 100 got removed

console.log("The Array after the shift operation is = ");

console.log(a);

console.log("The Deleted Value = ",a.shift()); // true can be removed

console.log("The Array after the shift operation is = ");

console.log(a);

a.shift('ABC','R','B');

console.log(a);

=========================================================

3) splice():

============

Syntax:

array-name.splice(index, count);

let a = [100,true,'A','R','B','JavaScript',2000,2024]

console.log("The Given array = ");

console.log(a);

a.splice(2,0);

console.log(a);

a.splice(2,3);

console.log(a);

====================================================================

Sorting of array elements:

==========================

Ecommerce application:

cart

at 6am ==> p1

at 6.28 am ==> p2

at 7.30 ==> p3

at 7.56 ==> p4

==> sorting ===> arranging of elements in ascending or descending order.

1) sort():

=========

==> sorting in ascending order by default.

Syntax:

array-name.sort()

2) reverse():

==============

==> to reverse the array.

Syntax:

array-name.reverse()

let a = [100,true,'A','R','B','JavaScript',2000,2024]

console.log("The Given array = ");

console.log(a);

a.sort()

console.log("The Array after the sorting is (Ascending Order)= ");

console.log(a);

(a.sort()).reverse();

console.log("The Array after the sorting is (Descending Order)= ");

console.log(a);

Assignment:

===========

WAP in javascript, to add new element into the dropdown and also remove the first dropdown option.

Ex:

Dropdown:

Electronics

Fashion

Grosessories

Beauty products

into this:

want to add:

Laptops & Mobiles

and remove:

Electronics

=============================================================================

**Day46: Object Type Part-01:**

Object Type

===========

number ==> numerals ==> 121, 1.02 etc.

string ==> '' or "" or ``

Boolean ==> true or false

undefined

null

arrays ==> [] or new Array()

Ex: a = [1,2,3,4,5,6]

log(a) ==> [1,2,3,4,5,6]

===========================

Form Registration:

==================

Name

Mail

Mobile

Password

application =====================> database

API

Name: ravi

Mail: ravivraoinfs@gmail.com

Mobile: 8977029779

Password: Ravi123

request: google.com

response:

after the registration

we can allow for login

at the time of login:

username/mail/mobile:admin123

password: admin123

application =========> database

API/Sql

==> Object type is a collection datatype

is the collection of elements which can be defined with key and value pair format.

Syntax for key value pair:

key-name : corresponding-value

==================================================

Creation of object type:

========================

==> four ways:

1) Using object literal:

========================

Syntax:

let/var/const object-name = {key1:val1,key2:val2,key3:val3,...};

const book = {title:'To Kill a Mockingbird',

author:"Harper Lee",

yearOfPublished:1960,

genres : ['Fiction','Drama'],

availableCopies:0,

isAvailable:function(){

return this.availableCopies > 0;

},

borrow:function(){

if(this.isAvailable()){

this.availableCopies--;

console.log("The Number of books remaining are = ",this.availableCopies);

}

else{

console.log("The Book is not available.");

}

}

};

console.log(typeof book);

console.log(book);

console.log(book.title);

console.log(book.author);

console.log(book.yearOfPublished);

console.log(book.genres);

console.log(book.availableCopies);

console.log(book.isAvailable());

book.borrow();

=========================================================

2) using Object() constructor:

==============================

Syntax:

let/var/const object-name = new Object();

Syntax for adding properties into an object:

object-name.property-name = value;

const car = new Object();

console.log(typeof car);

console.log(car);

// adding properties into an object

car.make = "Toyota";

car.model = "Corolla";

car.year = 2023;

console.log(car);

===============================================

3) Using class:

===============

class Car{

constructor(make,model,year,color){

this.make = make;

this.model = model;

this.year = year;

this.color = color;

}

describe(){

return `${this.year} ${this.color} ${this.make} ${this.model}`;

}

startEngine(){

return `${this.describe()} is starting...`;

}

stopEngine(){

return `${this.describe()} is stopping...`;

}

}

const car1 = new Car("Tesla","Model3",2024,"red");

console.log(car1.describe());

console.log(car1.startEngine());

console.log(car1.stopEngine());

**Day47: Conditional Statements:**

Conditional Statements

=======================

Sequential Execution:

=====================

==> Statement after the statement

Ex:

statement-1

statement-2

statement-3

Why conditional statements?

===========================

to make execute the application based on the selection we can use "conditional statements".

Types of Conditional Statements:

=================================

==> different types:

1) simple if statement

2) if-else statement

3) nested if else statement

4) if else if else ladder statement

5) switch statement

here, all the above statements must be work according to the condition.

condition ==> Relational operators (<, >, <=, >=, ==, !=)

1) simple if statement

======================

Syntax:

if(condition)

{

// if-block statements

statement-1;

statement-2;

}

==> if the condition is "true" then, the control can execute the statements in if-block. If the condition is "false", there is no output from the given program/application.

==> To overcome this, if we can define a statement after the if block:

in this case:

if the condition is "false" then, the control can execute the statement after the if block.

But when the condition is "true" then, the control can execute the program as normal (sequential execution).

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let age = prompt("Provide the age of the person:");

if(age >= 18){

document.write("He/She is eligible to Vote.");

}

document.write("He/She is not Eligible to Vote.");

</script>

</body>

</html>

=====================================

2) if-else statement

====================

Syntax:

if(condition)

{

// logic for if block

}

else{

//logic for else block

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let age = prompt("Provide the age of the person:");

if(age >= 18){

document.write("He/She is eligible to Vote.");

}

else{

document.write("He/She is not Eligible to Vote.");

}

</script>

</body>

</html>

===========================================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script>

let balance = 2345.0;

function depositFunds(){

let message = document.getElementById("message");

let amount = parseInt(prompt("Enter amount to deposit:"));

if(amount > 0){

balance += amount;

message.innerHTML = `Deposit Successful and Your account balance is = ${balance}`;

}

else{

message.innerHTML = 'Deposit Failure with negative funds.';

}

}

</script>

</head>

<body>

<h1> XYZ Bank </h1>

<button onclick = "depositFunds()"> Submit </button>

<div id = "message">

</div>

</body>

</html>

=====================================================

3) nested if else statement

===========================

Syntax:

======

if(condition1)

{

//if block

// statements

if(condition2)

{

//logic

}

else{

//logic

}

}

else{

if(condition3)

{

//logic

}

else{

//logic

}

}

a = 12

b = 21

c = 11

if(a > b){

if(a > c){

console.log("a is biggest.");

}

else{

console.log("c is biggest.");

}

}

else{

if(b > c){

console.log("b is biggest.");

}

else{

console.log("c is biggest.");

}

}

================================================

4) if else if else ladder statement

====================================

Syntax:

if(condition1){

//logic

}

else if(condition2)

{

//logic

}

else if(condition3){

//logic

}

else{

//logic

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let a,b,c,d,e;

a = parseInt(prompt("Enter a value:")); // 10

b = parseInt(prompt("Enter a value:")); // -9

c = parseInt(prompt("Enter a value:")); // 97

d = parseInt(prompt("Enter a value:")); // 79

e = parseInt(prompt("Enter a value:")); // -1

// a = 10;

// b = -9;

// c = 97;

// d = 79;

// e = -1;

if(a < b && a < c && a < d && a < e){

document.write("a is smallest number.<br>");

}

else if(b < c && b < d && b < e){

document.write("b is smallest number.<br>");

}

else if(c < d && c < e){

document.write("c is smallest number.<br>");

}

else if(d < e){

document.write("d is smallest number.<br>");

}

else{

document.write("e is smallest number.<br>");

}

</script>

</body>

</html>

Assignment:

===========

1) WAP IN JAVASCRIPT TO FIND THE BIGGEST NUMBER AMONG THE TWO NUMBERS.

2) WAP IN JAVASCRIPT TO FIND WHETHER THE GIVEN NUMBER IS POSITIVE OR NEGATIVE.

3) WAP IN JAVASCRIPT TO CHECK WHETHER THE GIVEN NUMBER IS EVEN OR ODD.

**Day48: Transfer And Switch Statements:**

Transfer Statements:

====================

==> two types:

1) break

========

==> is a keyword in JavaScript

==> can always use in loop body

for getting the immediate termination/exit from the loop.

Syntax:

break;

for(var i = 1;i <= 10;++i){

if(i == 5){

break;

}

console.log(i);

}

console.log("Hi");

console.log("Completed!");

=====================================

for(var i = 1;i <= 10;++i){

// break;

console.log(i);

break;

}

console.log("Hi");

console.log("Completed!");

===========================================================

2) continue

===========

==> a keyword

can be used in loops

to skip the current iteration and continue

with remaining statements.

Syntax:

continue;

for(var i = 1;i <= 10;++i){

if(i == 5){

continue;

}

console.log(i);

}

console.log("Hi");

console.log("Completed!");

var i = 20;

while(i > 0){

if(i == 10){

i -= 2;

continue;

}

console.log(i);

i -= 2;

}

Switch Statement:

=================

==> switch ===> Keyword

==> we can use to define the selection statement

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let s1,s2,s3,s4,s5;

s1 = parseInt(prompt("Enter marks:"));

s2 = parseInt(prompt("Enter marks:"));

s3 = parseInt(prompt("Enter marks:"));

s4 = parseInt(prompt("Enter marks:"));

s5 = parseInt(prompt("Enter marks:"));

total\_marks = s1 + s2 + s3 + s4 + s5;

percentage = total\_marks / 5;

document.write("The Percentage of the student = ",percentage,"<br>");

document.write("The Student Grade based on the percentage is = <br>");

if(percentage >= 85){

grade = 'A1';

}

else if(percentage >= 75 && percentage < 85){

grade = 'A';

}

else if(grade >= 70 && percentage < 75){

grade = "B";

}

else if(percentage >= 60 && percentage < 70){

grade = "C";

}

else if(percentage >= 50 && percentage < 60){

grade = "D";

}

else if(percentage >= 40 && percentage < 50){

grade = "E";

}

else{

grade = "F";

}

document.write(grade);

</script>

</body>

</html>

Syntax for the switch:

======================

switch(expression){

case label:

statements;

break;

case label:

statements;

break;

default:

statements;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

let m1,m2,m3,m4,m5;

m1 = parseInt(prompt("Enter marks:"));

m2 = parseInt(prompt("Enter marks:"));

m3 = parseInt(prompt("Enter marks:"));

m4 = parseInt(prompt("Enter marks:"));

m5 = parseInt(prompt("Enter marks:"));

let totalMarks = m1+m2+m3+m4+m5;

let percentage = totalMarks/5;

document.write("The Total Marks = "+totalMarks+"<br>");

document.write("The Percentage = "+percentage+"<br>");

document.write("The Grade of the student = <br>");

switch(true){

case percentage >= 90:document.write("A1");break;

case percentage >= 80:document.write("A2");break;

case percentage >= 75: document.write("A");break;

case percentage >= 65:document.write("B");break;

case percentage >= 55:document.write("C");break;

case percentage >= 40:document.write("D");break;

default:grade = "F";document.write("F");

}

</script>

</body>

</html>

**Day49: Objects Part\_02:**

Object Creation using create():

==============================

Syntax:

Object.create()

const Vehicle = {

type : 'Vehicle',

startEngine: function(){

console.log('Engine Started!');

},

stopEngine: function(){

console.log("Engine Stopped!");

}

};

console.log(Vehicle.type)

const Car = Object.create(Vehicle);

Car.type = "Car"; // existing property

Car.numWheels = 4; // new property

Car.drive = function(){

console.log("Driving the car in automatic mode...");

}

console.log(Car);

Car.drive();

Car.startEngine();

Car.stopEngine();

=======================================================

Accessing Object Properties:

============================

1) using . operator:

====================

Syntax:

Object-name.property-name

2) using []:

===========

Syntax:

Object-name['property-name']

3) using Iterators:

====================

for in:

======

==> used to access the property names from the given object.

Syntax:

for(var obj in Object-name){

//logic

}

for of:

======

==> not possible to access values

const Employee = {

name : "Jatin",

type : "Full-time",

shift : "ganeral-shift",

salary : 67785.0,

project:[

{client:"Samsung",domain:"Telecom",location:"Bangalore"},

{chipset:"Quad2553",product:"Qualcom"}

]

}

// Access the object properties

console.log(Employee.name);

console.log(Employee.type);

console.log(Employee.shift);

console.log(Employee.salary);

//using []

console.log(Employee['name']);

console.log(Employee['type']);

console.log(Employee['shift']);

console.log(Employee['salary']);

// for in iterator

for(let obj in Employee){

console.log(obj);

}

========================================

// Ecommerce Order

const order = {

orderId : 12345,

customer: {

name : "Manikanta",

email : "mani1122@gmail.com",

address : "1-1-11, XYZ Street, USA"

},

items : [{name:"Laptop",price:39999.0,quantity:1},{name:"Mouse", price: 2000.0,quantity:2},

{name:"Hoody", price: 1999,quantity:3}],

isPaid : true

}

order.items.forEach(item=>{

console.log(`Item:${item.name}, Price:${item.price} and Quantity:${item.quantity}`);

});

=====================================

ADDING NEW PROPERTIES INTO AN OBJECT:

=====================================

Syntax:

Object-name.newPropertyName = value;

DELETING PROPERTIES FROM OBJECT

===============================

Syntax:

delete objectName.PropertyName;

OBJECT PROPERTY MODIFICATION:

==============================

Syntax:

Object-name.propertyName = new-value;

const Student = {

name : "Rakesh",

age : 17

}

// modify property value

console.log(Student)

Student.name = "Kalyan";

Student.age = 18;

console.log(Student);

// adding new properties

Student.roll = 10;

Student.subject = "Maths";

console.log(Student)

// deleting the object property

delete Student.age;

console.log(Student);