SQL

SELECT colum\_name FROM table\_name

ex - SELECT c1,c2,..cn FROM table\_1(number);

SELECT \* FROM table\_1 - it will select the complete table

SELECT DISTINCT column FROM table

ex - SELECT DISTINCT(release\_year FROM film; -> will show unique years

COUNT -

SELECT COUNT(\*) FROM payment -> will show the number of rows in payment

SELECT COUNT(amount) FROM payment

SELECT COUNT(DISTINCT(amount)) FROM payment -> will show unique count

SELECT name,choice FROM table

WHERE name='person\_name' AND choice ='color\_name'; -> will return data with matching condition

SELECT COUNT(title) FROM film

WHERE rental\_rate >4 AND replacement\_cost >= 19.99

AND rating = 'R';

ORDER BY -

SELECT column\_1, column\_2 FROM table ORDER BY column\_1 ASC/DESC;

Spring

TOOLS -

SPRING BOOT , SPRING CLOUD , KUBERNETES , DOCKER

cONCEPTS -

Rest APi, Service Discovery, APi Gateway, Naming Server, Circuit Breaker

Load Balancer, Security

DIVERSE DEPLOYMENT OPTIONS -

Containers, Orchestration, Cloud native env

REST API -----> MICRO\_SERVICE ---> DOCKER ----> KUBERNETES

(spring boot) (SPRING CLOUD) (Containers) (Devops)

What is a web service - An application with

web layer

Business layer

Data layer

DB

Lets say an interviewr says - What if I create & share a JAR file for a web service

and ask will it work and be considered a web service

ans - A web service would require other dependencies for example DB dependencies

and a process to handle future updates of code

and what if it is not platform independent then it will not be considered a web service

A software designed to support platform independent ,interoperable machine to machine interaction

over a network is called web service

Making Web service platform independent -

the communication (req and response) should be platform independent

popular ones are XML or JSON

Web service terminology -

Request - input to a web service

Response - output from the web service

Message exchange format - format used for communication

Service provider (or server) - entity providing the web service

Service Consumer - entity consuming the web service

SOAP VS REST WEB Service -

In SOAP web service using communication format is SOAP XML

it conatins SOAP ENV , SOAP Header, SOAP body

In REST web service communication format is HTTP using GET,POST,DELETE for executing operations

SPRING BOOT -

World before Spring boot -

MANAGE FRAMEWORK AND VERSIONS / DEPENDENCY MANGMENT (POM XML)

- Rest API - spring framework, spring MVC framework, JSON binding framework

- Unit tests - spring test, Mockito, Junit

WEB.XML - if we wanted to use spring MVC, we would configure dispatcher servelet

SPRING CONFIG - if we are building a web application we will need to define

component scan, view resolver

if it was DB application we would define data soruce config

Manage Spring Beans (context.xml)

LOGGING , ERROR HANDLING, MONITORING

Spring Web Web

Build web, including RESTful, applications using Spring MVC.

Uses Apache Tomcat as the default embedded container.

Spring Boot Starter Projects

- we need a lot of framework to build application features:

- web & REST API -Spring boot starter web, Spring MVC, Tomact, JSON conversion

- writing Unit tests -Spring starter Test, Junit, Mockito,..

- Databse - Spring boot starter data JPA

Spring boot starter data JDBC

- Security - Spring Boot Starter Security

Spring starter dependency groups them to make it easy to build application

Spring boot devtools -

add dependency in the pom file

and change the java file , save , reload the browser and changes will be reflected

remembver - For pom.xml dependency changes,you will need to restart server manually

Managing App. Config using profiles

Application have diff environments: Dev, QA, Stage , Prod

diff env neeeds diff config

different Database

different Web services

Example -

we will have diff application property files (logging leveles -trace> debug> info> warning> error> off)

application-dev.properties - logging level = trace

applciation-prod.properties - logging level = info

and in applciation.properties - spring.profiles.active = prod

Sring boot -

Annotations -

@Component - If we want a class to be a Spring bean we use @Component

@Service @Repository @Controller - these 3 are also called as stereotype annotations

Service marks the class as a business layer class

Repository marks the class as a Data Layer which typically contains CURD methos

Controller allows inbuilt Spring MVC framework to access the class as a Web Request Handler

@Autowired - to mark a dependency which Spring is going to resolve and inject. We can use this annotation with a constructor, setter, or field injection.

this enables loose coupling

@Value - (Injects with a fixed value) we can set a value from a property file

if a file has a key value pair, we can use the key to find the fiel & inject value

Ex - @Value("${database.url}")

it is typically used with @bean or stereotype annotation using SPEL(spring expression language)

this enable loose coupling

@Configuration -

@Bean -

marks a factory method which instantiates a Spring bean:

Spring calls these methods when a new instance of the return type is required.

The resulting bean has the same name as the factory method. If we want to name it differently, we can do so with the name,

or the value arguments of this annotation (the argument value is an alias for the argument name):

Note, that all methods annotated with @Bean must be in @Configuration classes.

@AutoConfiguration -

logging.level.org.springframework=debug if we log this in applciation property

we will see that springboot has auto configured in the jar file

dispatcher servelet

embedded servelet container

error mvc auto configuration

Bean <--> JSON (jsonHTTPmessageConverterConfig)

Embedded Servers - JAR(Embedded server - Tomcat)

step 1 - install Java

step 2 - run JAR file

Actuator in Spring Boot -

->Monitor and manage your application in your production

->provides a number of endpoints

-beans - Complete list of Spring beans in your app

-health - Applicaiton health information

-metrics - Application metrics

-mapping - Details around Request Mapppings

dependency - spring-boot-starter-actuator

Applicaiton properties for actuator -

by default it will only expose the health end point

for additonal end points use

management.endpoints.web.exposure.include=\*

more actuator means more memory useage

Spring vs Spring MVC vs Spring boot

- Spring framework : dependency injection

@Component, @Autowired, Component Scan etc

also provides Spring modules, Spring projects for other framework integration

ex - hibernate / JPA , junit & mockito

- Spring MVC: Simplify building web apps and REST APi

Building web application is easy with @Controller, @RequestMapping

-Spring boot - build prod ready app quickly

Auto configuration

Enable non functional requirments -

Actuator, Embedded servers, LOGGING, Profiles and Configuration

JUNIT -

Testing - check the app behaviour against expected behaviour

option 1 - Deploy the complete application and test know as system testing or integration testing

option 2 - Test specific unit of application code independently know as unit testing

example -

Class {

method1 = calculateSquare(x) = x\*x;

method2;

method3;

}

test case -

calculateSquare(5) = 25

calculateSquare(25) = 625

Junit may not run unit test in order, it may run test 2 first and then test 1

so we use

@BeforeEach annotation if we want run a test before test 1,2

similarly @AfterEach is present

@BeforeAll - would run before all the test cases and similarly @AfterAll

syntax - @BeforeAll

static void beforeAll

syso

overall output -

beforeAll

BeforeEach

test1

AfterEach

BeforeEach

test2

AfterEach

afterAll

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

MAVEN and GRADLE

Maven - Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM),

Maven can manage a project's build, reporting and documentation from a central piece of information.

Things you do when you code

- create new projects

-Manage dependencies

-Build JAR

-Run your JAR locally in Tomcat

-Run unit tests

-Deploy in test env

Maven helps you do all these

Maven cammands

mvn --version -> shows the maven version

mvn compile - compile source files

mvn test compile

mvn clean - clean the target folder

mvn test - will run the unit test

mvn install

mvn package - create a jar file

mvn help - effective pom

mvn dependency:tree

HTML CSS JS

How the Internet Actually works

Client -> IPS - > DNS -> send back the IP ADD ->Client -> Website -> Data -> Client

Website usually works with HTML, CSS, JS files

HTML -> Contains the Content

CSS -> Used for Styling

JS file -> Makes the website dynamic

HTML ->

Tags vs Elements

<h1> Hello </h1>

here h1 is the tag whereas the complete line is the the element

Heading Tag -

<h1> Heading 1 </h1>

..

...

.....

<h6> Heading 6</h6>

Paragraph element - used to mark paragraphs

<p> This is a paragraph </p>

Horizontal Rule element -

<hr/> it is also called a void element

Break element - </br>

Defines a single line break

Unordered List -

<ul> </ul>

<li> -> Defines a list item </li>

example -

<ul>

<li> Milk </li>

<li> Egg </li>

<li> Water </li>

<li> Bread </li>

o/p will be

\*Milk

\*Bread

\*Egg

\*Water

As the order of the list dosent matter we use ul tag

Ordered List -

<ol>

<li> Egg </li>

<li> Water </li>

<li> Bread </li>

</ol>

o/p -

1.Egg

2.Water

3.Bread

Nesting and Indentation -

<ul>

<li>

<ul>

<li></li>

</ul>

</li>

<li> Start Work</li>

</ul>

o/p

\*Wake up

\*Drink Water

\*Make Bread

\*Break Egg

\*Mix Flour

\*Bake

\*Start Work

This is called Nested List and Indentation is making the tag readable in code

Anchor Elements -

<a>this is a link</a>

<a href = "https">this is a link</a>

<tag attribute = value > content </tag>

Global Attributes ->

attributes which can be used on any html element

example -

<a dragable = true> this is a link </a>

image tag -

<img src ="url" />

src is the attribute which tells the location of the image also img is a void tag

example - <img src="https://picsum.photos/400" />

alt attribute for image tag - alternate text description

Relative file paths - helps the html file to navigate media item using ../

instead of absolute path

go up level in file path - ../Rootfolder.docx

current directory - ./dog.png or dog.png

below in file path - images/cat.png

multi page websites -

<a href ="https:/">About Page</a>

using achor tag we can navigate between webpages

HTML Boiler-plate -

this allows to understand the HTML file structure

<!DOCTYPE html> -->tells browser that file is written in html

<html lang ="en"> --> sets the language of the text content in the file

<head> --> encapsulation (dosent show the user the info)

<meta charset ="UTF-8"> --> ensures the characters displayed correctly

<title>Google</title> --> shows the name in the website bar

</head>

<body>

//content of the website - text, title, img src, links

ex - <h1> My website </h1>

</body>

</html>

short cut for vs code - shift+1 after creating html file

Hosting Your Website -

github -> create new repo ->name html-portfolio -> Add public -> Add Readme file

upload the html and css file with the index.html name(not the folder but the file)

commit changes -> setting -> pages -> branch to main

refresh and you will see a pop up

CSS - cascading style sheet

Inline, internal, external are the 3 ways of adding css

Inline css is added to the html tag directly

exmaple - <html style ="background:blue"> </html>

useful to style single element

Internal css uses style tag to add css element

example -

html

head

style

html{

background: red;

}

style

head

body

body

html

Internal style css are useful for one html file not for multipaage website

External Css - it is written in a different file and is used

to connect with the html file using <link> tag with rel = "stylesheet"

href ="" attributes

CSS selectors -

its the part that selects the html tag

ex - h1{

color:blue

} here h1 is the css selector

Class selector -

.red-text{

color:red

} here we use . and the name of the class

and we specify the class in html

ex - <h2 class="red-text">Heading 2</h2> - o/p will be in red color

<h2>Heading 3</h2> - o/p will be in black color

ID selector -

#main{

color:red

} here we use # and main (id name)

and we specify in html

ex - <h2 class="red-text">Heading 2</h2> - o/p will be in red color

<h2 id ="main">Heading 2</h2> - o/p will be in red color

<h2>Heading 3</h2> - o/p will be in black color

the difference between class selector and id selector is that

class selector can be applied to many elements in html file

but id selector can be applied to only one element in html file

Attribute selector -

p[draggable]{

color: red

} here p is the html element and draggable is the attribute

what this means is select all p elements with attribute draggable

and apply the css style to it

ex -

<p draggable="true">Drag me</p> -o/p will be in red color

<p>Dont drag me</p> -o/p will be in black color

universal selector -

\*{

color:red

} - it will apply to the complete CSS file

font size properties of CSS

font-weight - sets how thick or thin characters in text should be displayed.

normal bold , lighter/bolder -relative to parent , number - 100 to 900

font size - sets the size 1px - 1/96th inch - 0.26mm

1 pt(point) - 1/72 inch - 0.35mm

1 em(m) - sets the size to 100% of parent

1 rem - sets the size to 100% of root

font family - sets to the font family

using link rel = "" href ="" we can copy the google fonts to our file

example

#cursive{

font-family - cursive;

}

<body>

<p id = cursive> font paragraph </p>

</body>

text-align - aligns the text to position mentioned

Box Model of CSS -

height - sets the height of the box

width - sets the width of the box

border - we can set 3 values (thickness) (style of border) (color of the border)

example - 10px solid/dashed black

border-width - set width for all the sides of border

example - 0px 10px 20px 30px- sets for top right bottom left relativly

10px 20px - sets for top+bottom and left+right

Padding - used to create space around an element's content, inside of any defined borders.

Margin - used to create space around elements, outside of any defined borders.

in a website it would look like

Margin

|

Border

|

Padding

|

element of html

|

Padding

|

Border

|

Margin

Content Divison Element - seperates the content in a file

<div></div>

Cascading in CSS - multiple rules can be applied in css but the order of importance

when you finish the cascade will be applied

html - css -

ex - <ol> li{ color: green}

<li>One</li>

<li style="color:red">Two</li>

<li>Three</li>

</ol>

o/p - > one and three in green color , two in red color

order of importance -

according to position - lower the css element will be applied

ex - li{

color:red;

color:blue;

}

color blue will be applied as it is lower

according to Specificity - the order would be

element

class

attribute

id

which means id>attribute>class>element

ex - li{ //li is the element

color:blue

}

.first-class //class

{

color: red;

}

li[draggable] //attribute draggable

{

color:purple;

}

#first-id{ //id is #first

color:orange;

}

in html

<li id="first-id" class="first-class" draggable>

o/p of the text would be in orange

according to type - the order would be inline > internal > external

ex -

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

<style></style> //internal

<h1 style =" ">Helloo</h1> //inline

according to importance - using a keyword !important

we can ensure that the rule will be selected

Combining CSS selectors-

html file

<body>

<h1> Todo list </h1>

<h2>Monday</h2>

<div class = "box">

<p class="done"> Do these things today!</p>

<ul class="list">

<li>Wash Clothes</li>

<li class = "done"> Read</li>

<li class="done"> Maths question</li>

</ul>

</div>

<ul>

<p class ="done">Other items</p>

</ul>

<p>the best thing todo is</p>

<body>

</html>

we can combine css selectors using a

Group - apply to both selectors-

ex -

h1, h2{

color: bluevoilet;

}

Child - selecting parent and direct child(meaning directly nested)elements together

ex -

.box > p { //box is the parent and p is the direct child

color : firebrick;

}

Descendant - Apply to a descendent of left side

.box(space)li{

color : blue;

}

chaining - apply where ALL selectors are true

li.done{

color: sea;

}

combining combiners -

ul p.done{

font size:0.5rem;

}

CSS positions -

Static

- In Static the element is positioned according to the Normal Flow of the document.

The top, right, bottom, left, and z-index properties have no effect. This is the default value.

Relative

The element is positioned according to the normal flow of the document, and then offset relative to itself

based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements;

thus, the space given for the element in the page layout is the same as if position were static.

This value creates a new stacking context when the value of z-index is not auto.

Its effect on table-\*-group, table-row, table-column, table-cell, and table-caption elements is undefined.

absolute

The element is removed from the normal document flow, and no space is created for the element in the page layout.

The element is positioned relative to its closest positioned ancestor (if any) or to the initial containing block.

Its final position is determined by the values of top, right, bottom, and left.

This value creates a new stacking context when the value of z-index is not auto.

The margins of absolutely positioned boxes do not collapse with other margins.

fixed

The element is removed from the normal document flow, and no space is created for the element in the page layout.

The element is positioned relative to its initial containing block, which is the viewport in the case of visual media.

Its final position is determined by the values of top, right, bottom, and left. This value always creates a new stacking context.

In printed documents, the element is placed in the same position on every page.

sticky

The element is positioned according to the normal flow of the document, and then offset relative to its nearest scrolling ancestor and containing block

(nearest block-level ancestor), including table-related elements, based on the values of top, right, bottom, and left.

The offset does not affect the position of any other elements. This value always creates a new stacking context. Note that a sticky element "sticks"

to its nearest ancestor that has a "scrolling mechanism" (created when overflow is hidden, scroll, auto, or overlay), even if that ancestor isn't the nearest actually scrolling ancestor.

CSS DISPLAY -

Ex - <p>Hello<span>Beautiful</span>world</p>

Here each element has its own display property value

for span tag the default value of display - inline

for most of the elements display - block

if the display is block it will take the complete horizontal width

example - Hello

WORLD

block element allows us to set height and width

if it is inline - the elements will go in the same horizontal width until width runs out

example - Hello World

we cannot set the width and height for inline, it takes the space required to display the HELLO WORLD

by combining both we get inline-block element - which allows to set height and width and keeps the element side by side

CSS FLOAT -

If we wanted our text to wrap around an inline-block element then we would use float property

example -

\_\_\_\_\_\_\_\_\_\_\_\_

|inline | loreum ipusumloreum ipusum

|block | loreum ipusumloreum ipusum

|\_\_\_\_\_\_\_\_\_\_\_\_| loreum ipusumloreum ipusum

loreumipusumloreumipusumloreum ipusumloreum

loreumipusumloreumipusumloreum ipusumloreum

loreumipusumloreumipusumloreum ipusumloreum

Html - <img .../>

<p>loreum ipusum</p>

css - img{

float - right or left;

}

Clear property - The clear CSS property sets whether an element must be moved below (cleared) floating elements that precede it.

The clear property applies to floating and non-floating elements.

When applied to non-floating blocks, it moves the border edge of the element down until it is below the margin edge of all relevant floats.

The non-floated block's top margin collapses.

RESPONSIVE Website - to make the website responsive we use

1 - Media Queries

@media(max-width: 600px){ //if the screen size is 600px css properties will be applied

/\* CSS for screens below or equal to 600px wide \*/

}

combining Media queries

@media(min-width:600px) and (max-width:900px){

//css

}

this property will set the size from 600 to 900

@media(min-width:900px) and (max-width:600px){

//css

}

this will include anything that less than 600px and anything that is more than 900px

Screen property -

@media screen(oreientation: landscape){

//css

}

this will set the screen oreientation

2 - CSS Grid

this sets the display to various grid option layout. this is mainly used for 2d layout

where we can define how to lay out the coloum and rows

example -

html

<div class ="grid-container">

<div class = "first-card"></div>

<div class = "card"></div>

<div class = "card"></div>

<div class = "card"></div>

<div class = "card"></div>

</div>

css -

.grid-container{

display : grid; //can be grid, inline, block , flex

grid-template-columns: 1fr 1fr; //we want two columns 1 and 2 / fr stands for equal columns

grid-template-rows: 100px 200px 300px; //sets the height of the rows in the columns

gap: 30px; // gap between the rows

}

.first{

grid-column: span 2; // this sets the first column to full width by setting it to span 2 columns

}

.card{

background-color:blue;

}

3 - CSS Flexbox -

flexbox allows us to create 1d layout be it horizontal or vertical

example -

html -

<div class ="flex-container">

<div class = "first card"></div>

<div class = "second card"></div>

<div class = "card"></div>

<div class = "card"></div>

css -

.flex-container{

display : flex; //can be grid, inline, block , flex

}

.first{

flex: 2;//this sets the first column to width/height 2:1

}

.card{

background-color:blue;

border: 30px solid white;

height: 100px;

flex: 1;

}

.second{

flex: 0.5;

}

Note -

Flexbox will apply flexbox rules in to the elements present inside it i.e inline

ex - if we have 4 elements div (block), span(inline) , p(block), img(inline) inside flexbox

all the elements will be displayed inline for flexbox

Flex Direction -

by default this direction property is set to row i.e left to right (main axis) and top to bottom(cross axis)

if

flex direction: column then top to bottom(main axis) and left to right(cross axis)

flex basis - sets the height for coloum and width for the row

4 - External Framework (bootstrap)

bootstrap offers pre built css, only the class needs to be mentioned by the user

example -

<div class = "container">

<div class ="row">

<div class ="card col-6"></div>

</div>

</div>

bootstrap is built on top of flex box, and divides into 12 columns

JAVASCRIPT -

Role of js in web dev is to add dynamic effects and interactivity

web application frameworks of JAVASCRIPT - React , angular

web application on web servers - node js

value and variables -

console.log("Jones") -> jones is the value

let js = "amazing" -> variable

variable declaration - first , firstName, firstNamePerson , first\_name\_person ;

camel case notation

Datatypes -

Number - floating point numbers (used for decimals and integers)

String - Sequence of characters

Boolean - Logical type that can only be true or false

javascript has dynamic typing - we dont have to manually define the dataype

of value stored in a variable.

ex - let js = 23 ;

typeof js

js = "SIKE" ;

typeof js

o/p will be number and string

similarly if we say

let js;

console.log(typeof js)

o/p will be undefined

declaring variable using let, const, var

let allows the variable to be changed later for re-assigning the variable or declaring empty variables declaration

const dosent allow the variables to change also called immutable and dosent also allows empty variable declaration

var is same as let but was used in older versions

Read basic operators

Operator precedence - 1 - highest , 13 - lowest

1) grouping - (...)

2) member access, new(with argument list), function call, optional chaining

3) new(without argument list)

4) postfix increment, postfix decrement

5) logical operators , type of , void ,delete, await

6) exponentital

7) multiplication , divison ,

8) add , sub

9) bitwise

10) less than , greater than

11) equality , inquality

12) conditional

13) assignment

String and Template literals - we can replace $({variable}) to create a string line

if/else -

const age = 15;

if(age>=18){

console.log('Sarah can drive'); }

else{

const yearLeaft = 18 - age;

console.log('Sarah is young wait another $(yearLeaft) years;);

}

type conversion -

const inputYear = '1991';

console.log(Number(inputYear)); //converting string to number

console.log(inputYear + 18); //this is still a string to convert it to a number

we change it to console.log(Number(inputYear)+ 18);

NaN - not a number - means invalid number

we can only convert to number , string , boolean

type coercion - happens when operator is dealing with 2 different value, where js converts

one variable to match other

example - using template laterals

console.log('i am' + 23 + 'years old');

console.log('i am' + '23' + 'years old'); //both o/p will be same due to coercion

console.log('23' - '10' -3); //o/p is subtraction number

console.log('23'+10+3) //o/p is a 2313

console.log('23'/'2') //o/p is remainder

booleans - trutsy and falsy values -

we only have 5 falsy values: 0,'',undefined, null, NaN

=== & == operator

=== strict eqallity compares the value of the variable without type coercion

== loose eqallity compares the value with type coercion

example - '18' == 18 -> true // '18' === 18 -> false

boolean logic - using logical operators we can use boolean values

AND - true if both are true otherwise false

OR - true if anyone is true

NOT - inverts true/false value

Switch statement -

const day = 'Monday';

switch(day){

case 'Monday';

console.log('Plan course structure');

break;

case 'tuesday'

console.log('prep theory vids');

break;

default

console.log('Not a day');

}

terenay operator - is an expression of the loop it works best for quick decision

example - const age = 24;

const drink = age >= 18 ? 'wine' : 'water';

console.log(drink);

the above const drink statement is comparable to

if(age >=18){

drink = 'wine';}

else{ drink = 'water';}

example with a template literal -

console.log('I like to drink ${age >= 18 ? 'wine' : 'water}');

example -

Steven needs a very simple tip calculator for whenever he goes to eat in a restaurant.

In his country, it's usual to tip 15% if the bill value is between 50 and 300.

If the value is different, the tip is 20%.

solution -

const bill = 550;

const tip = bill >=50 && bill <=300 ? bill \* 0.15 : bill \* 0.2;

console.log(`The bill was ${bill}, the tip was ${tip}, and the total value ${bill + tip}`);

Strict Mode -

Helps in writing a sercure javascript code , it helps by cataching error

and display in the console , it also reservers the keywords which will be used in future releases

to activate strict mode

the very first statement shud be 'use strict';

example -

'use strict';

let hasDriversLicense - false;

const passTest = true ;

if(passTest) hasDriverLicense = true;

if(hasDriversLicense) console.log('I can drive :D');

error in console on using strict mode - uncaught reference error:

without using strict mode the console will not log this reference error

also using strict mode we cannot use keyword for variables

const if = 12; // not allowed

Functions -

A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value,

but for a procedure to qualify as a function, it should take some input and return an output

function are reusable code blocks

Syntax -

function logger(){

//function body

console.log('My name is jonas');

}

logger(); //invoking,calling,running the function

we can call this function as many times we want

example -

function fruitProcessor(apples, oranges)//function with variables

{

cosnt juice = 'Juice with $(apples) apples and $(orange) oranges.';

return juice;

}

const appleJuice = fruitProcessor(5,0); //storing the return variable

console.log(appleJuice);

function declaration vs function expression

What Are Function Declarations?

Function declarations are when you create a function and give it a name.

You declare the name of the function when you write the function keyword,

followed by the function name.

For example:

function logger(){

//function body

console.log('My name is jonas');

}

As you can see, the function name (myFunction) is declared when the function is created.

This means that you can call the function before it is defined.

What Are Function Expressions?

Function expressions are when you create a function and assign it to a variable.

The function is anonymous, which means it doesn’t have a name.

For example:

let add = function(a,b){

return a+b;

};

As you can see, the function is assigned to the myFunction variable. This means that you must define the function before you can call it.

Differences Between Function Expressions & Declarations

Function declarations are hoisted, while function expressions are not. This means that you can call a function declaration before it is defined,

but you cannot do this with a function expression.

With function expressions, you can use a function immediately after it is defined. With function declarations, you have to wait until the entire script has been parsed.

Function expressions can be used as an argument to another function, but function declarations cannot.

Function expressions can be anonymous, while function declarations cannot.

ARROW FUNCTION -

example -

//function declaration

const calAge2 = function(birthYear){

return 2024- birthYear;

}

//ARROW function

const calAge3 = birthYear => 2024 - birthYear;

const age3 = calAge3(1999);

console.log(age3);

//ARROW fn for more than one line code with multi parameters

const retirementYear = (birthYear, firstName) =>{

const age = 2024 - birthYear;

const retirement = 65- age ;

return '${firstName} retires in ${retirement} years';

}

console.log(retirementYear(1999, 'meme'));

console.log(retirementYear(1999, 'mene'));

Function calling other Function -

const cutPieces = function(fruit){

return fruit\*4;

}

const fruitProcessor = function(apples,oranges){

const applePieces = cutPieces(apples);

const orangePieces = cutPieces(oranges);

const juice = 'Juice with ${applePieces} pieces of apple

and ${orangePieces} pieces of orange.';

return juice ;

};

console.log(fruitProcessor(2,3);

ARRAYS -

example -

const friend 1 = 'Micheal';

const friend 2 = 'Michio';

const friend 3 = 'Mike';

const friends = [ 'Micheal', Michio', 'Mike']; //one way to store

console.log(friends);

const year = new Array(1991, 1999, 2000, 2020); // another way to store

console.log(friends[0]); //logging the first element

console.log(friends.length); //gives us the number of elements

console.log(friends[friends.length-1]); //leftmost value

friends[2] = 'Jay' //replaces the element from the array

//add element

const newLength = friends.push('Jay'); //adds to last

console.log(friends);

console.log(friends.length);

friends.unshift('John'); //adds to first

console.log(friends);

//Remove elements

friends.pop(); //removes last

const.log(popped);

console.log(firends);

friends.shift(); //first

console.log(friends);

console.log(friends.indexOf('steven'));// if the element is present it will give the index value

console.log(firends.indexOf('Bob'));// element is not present value return is '-1'

console.log(friends.includes('steven')) //returns true if element is present

console.log(friends.includes('bob')) // returns false

//includes uses strict equallity so it would return false if string is matched with number

frieds.push(23);

console.log(friends.includes('23')) //returns false

console.log(friends.includes(23)) //return true

if(friends.includes('Steven')){

console.log('You have a friend called Steven');

} // returns the console log statement

OBJECTS - in arrays we cannot reference the elements in the array by their name(can only reference in order)

so we use objects in objects we define key value pair, then we can give every value a name

in array the declaration order of the elements matters when we retrive them

in objects the declaration order of the elements dosent matter

example -

array -

const dbzArray = [

'Goku',

'Kakrot',

2025-1999,

'fighter',

['Vegeta', 'picoolo', 'kirlin']];

object literal syntax -

const dbz ={

firstName: 'Goku',

lastName: 'Kakrot',

age: 2025-1999,

job: 'fighter',

friends: ['Vegeta', 'picoolo', 'kirlin']};

console.log(dbz);

console.log(dbz.lastname); //dot notation

console.log(dbz['lastName']); //bracket notation

//bracket notation

const nameKey ='Name';

console.log(jonas['first'+ nameKey]);

console.log(jonas['last'+ nameKey]);

in dot notation we use the real final property name

in bracket we use a commputed property name

when we need to compute the property name we use bracket notation

prompt('What do you want to know about Jonas');

//storing the promt

const intrestedIn = prompt('What do you want to know about Jonas');

console.log(intrestedIn);

console o/p - selected element ex - job

so to access the element of the object

we use bracket notation

console.log(jonas[intrestedIn]);

lets add if else block to see if they access other options

if(jonas[interestedIn]) //here if block will check for the values

{

console.log(jonas[intrestedIn]);

}else{

console.log('Wrong req');

}

adding new properties to the object

dbz.location = 'Earth'; //adding using dot notation

dbz['twitter'] = 'super saiyan'; //adding using bracket notaiton

//add the following in object

// "Goku" has 3 friends, and his best friend is called '

console.log(`${dbz.firstName} has ${dbz.friends.length} firends, and his best friend is called ${dbz.firends[0]}');

OBJECTS METHODS - So we learned that objects just like arrays, can hold different types of data.

And they can hold even arrays, and in fact, they could even hold objects inside objects.

But now we can take it even further. And for that, remember how I said that functions are really just another type of value.

And if a function is just a value then that means that we can create a key value pair

in which the value is a function. And that then means that we can in fact, add functions to objects.

example -

const dbz =

{

firstName: 'Goku',

lastName: 'Kakrot',

birthYear: 1999,

job: 'fighter',

friends: ['Vegeta', 'picoolo', 'kirlin']

hasCar: false,

//adding a function expression

//calAge: function(birthYear){

//return 2024 - birthYear;

//}

//commenting out birthYear from the function to use this keyword

calAge: function(){

console.log(this);

return 2024 - this.birthYear;

},

//challenge to add a summary method which returns string

//Goku is a 25 year old fighter, and he has no driver's license.

getSummary:function(){

return `${this.firstName} is a ${this.calAge()} year old ${dbz.job}, and he has ${this.hasDriverLicense ? 'a':'no'} driver's license.`

}

};

console.log(dbz.calAge()); //using dot notation no need to pass argument bcuz of this keyword

//console.log(dbz['calAge'](1999)); //using bracket

//here we are passing birth year and also it is present as obj property

//this can cause error

//so we use this keyword to access from the object property

//if we need to use the calAge function many times then instead of passing

//it multiple times we can use this key word to store the new property

calAge: function(){

this.age = 2024 - this.birthYear;

return this.age;

}

console.log(dbz.age);

FOR LOOPS -

for(let rep = 1; rep<=10; rep++){

console.log(`Lift weights for ${rep}');

}

printing the array -

//instead of providing the length as i <5

//we use array.length property to print the array

const dbz = [

'Goku',

'Kakrot',

2025-1999,

'fighter',

['Vegeta', 'picoolo', 'kirlin'],

];

for(let i =0; i<dbz.length ;i++){

console.log(dbz[i]);

}

//using two arrays

const years =[1997, 1999,2000,2002];

const ages =[];

for(let i =0; i<years.length;i++){

ages.push(2024 - years[i]);

}

console.log(ages);

//continue and break

console.log('Print only STRING ----')

for(let i =0; i<dbz.length ;i++){

if(typeof dbz[i] !== 'string') continue;

console.log(dbz[i], typeof dbz[i]);

}

console.log(' break with Number----')

for(let i =0; i<dbz.length ;i++){

if(typeof dbz[i] == 'number') break;

console.log(dbz[i], typeof dbz[i]);

}

//looping backwards -

//it should print the array element [4,3,2,..0]

for(let i=array.length-1; i>=0 ; i--){

console.log(i,array[i]); }

loop inside loop -

for(let exercise = 1; exercise < 4; exercise++){

console.log(`----Starting Excersie ${exercise}`);

for(let rep =1 ; rep <8; rep ++){

console.log(`Excersie${exercise}: Lifting weights repetition ${rep}`);

}

}

while loop -

example - rolling a dice till we get number 6 using while loop

in this case we dont know how many times the loop will run, that is when while loop is used for

let dice = Math.trunc(Math.random()\*6)+1;

while(dice != 6){

console.log(`you rolled a ${dice}`);

let dice = Math.trunc(Math.random()\*6)+1;

}

CHALLENGE - YOUR TASKS:

Create an array called bills containing all 10 test bill values.

Create empty arrays for the tips and the totals (tips and totals)

Use the calcTip function we wrote before (included in the starter code) to calculate tips and total values (bill + tip)

for every bill value in the bills array. Use a for loop to perform the 10 calculations!

TEST DATA: 22, 295, 176, 440, 37, 105, 10, 1100, 86, and 52.

BONUS:

Write a function calcAverage which takes an array called arr as an argument. This function calculates the average of all numbers in the given array.

This is a DIFFICULT challenge (we haven't done this before)! Here is how to solve it if you feel like it:

First, you will need to add up all values in the array. To do the addition, start by creating a variable sum that starts at 0.

Then loop over the array using a for loop. In each iteration, add the current value to the sum variable. This way, by the end of the loop, you have all values added together.

To calculate the average, divide the sum you calculated before by the length of the array (because that's the number of elements).

Call the function with the totals array.

solution -

const calcTip = function (bill) {

return bill >= 50 && bill <= 300 ? bill \* 0.15 : bill \* 0.2;

}

const bills = [22, 295, 176, 440, 37, 105, 10, 1100, 86, 52];

const tips = [];

const totals = [];

for (let i = 0; i < bills.length; i++) {

const tip = calcTip(bills[i]);

tips.push(tip);

totals.push(tip + bills[i]);

}

console.log(bills, tips, totals);

const calcAverage = function (arr) {

let sum = 0;

for (let i = 0; i < arr.length; i++) {

// sum = sum + arr[i];

sum += arr[i];

}

return sum / arr.length;

}

console.log(calcAverage([2, 3, 7]));

console.log(calcAverage(totals));

console.log(calcAverage(tips));

4 step solve any problem -

1) make sure you understand 100% of the question.

2) divide and conquer - break big problem to smaller sub-problem

3) do research

4)write pseudo-code

DOM and DOM Manipulation -

document.querySelector('.elementname')//used to select element from html

console.log(document.querySelector('.elementname'));

example -

console.log(document.querySelector('.message').textContent);

document.querySelector('.message').textContent = '🎉 Correct Number!';

DOM Structure

DOCUMENT - special object that is the entry point to the DOM

ELEMENT - <html>

ELEMENT - <body>/<head>

ELEMENT - <section>/<title>

ELEMENT -<p>/<img>

TEXT

DOM methods and properties are part of WEB API -> which can interact with JS

HANDLING CLICKING EVENTS -

we use this to listen to a clicking events like buttons

so we use addEventListener using querySelector followed by type of the event and function which will store the value

syntax -

document.querySelector('.elementname').addEventListener('event type', function (){ function body })

example -

document.querySelector('.check').addEventListener('click', function (){ })

MODAL-WINDOW - manipulates classes , and esc keyword

if you want to use the same function in multiple event listeners, then you need to specify that function

as a separate function, like openModal function and close modal and then you can pass that function

as an argument to multiple "add event listener" methods. that's one of the takeaways

And the other one is that,use the functionality of adding and removing classes all the time

in order to change the appearance of elements on our page and that's because classes allow us

to basically aggregate multiple CSS properties in just one, like a container.

So each class functions a bit like a container with a lot of properties in it. and then here, by adding and removing them,

we basically can activate and deactivate certain styles, all at the same time.

adding escp btn functionality -

document.addEventListener('keydown', function(e)){

console.log(e)

console.log(e.key)

}

here addEventListener listens to all the events happening on the webpage and trigger the event

keydown is an event for pressing any key on keyboard

JavaScript will genrate an object and we can

console.log(e) will provide the o/p of all the keyboard events

ex -

KeyboardEvent {

isTrusted: true

altKey: false

bubbles: true

cancelBubble: false

cancelable: true

charCode: 0

code: "Escape"

composed: true

ctrlKey: false

currentTarget: null

defaultPrevented: false

detail: 0

eventPhase: 0

isComposing: false

key: "Escape"

keyCode: 27

location: 0

metaKey: false

repeat: false

returnValue: true

shiftKey: false

}

CONCURRENCY MODEL of JAVASCRIPT -

javascript engine handles multiple tasks happening at the same time

why we need that ?

JS runs in one single thread, so it can only do one thing at a time.

So what about a long running task ?

By using an event loop : takes long running tasks, executes them in the bg

and puts them back in the main thread once they are finished

JS Engine - program that executes JS code

example - google V8 engine

js engine has two memory

call stack (where our code is executed) | heap(where object are stored)

contains execution context | contains object in memory

JavaScript used to be interpreter languange but was converted to

JIT ( just in time ) compilation : Entire code is converted into machine code at once, then executed immediately

Modern JIT JS engine workflow -

CODE -> Parsing -> Compilation -> Execution -> Optimization -> Compilation -> Execution

by optimizing an unoptimized code while simultaneously running the code JS ensures code is running fast

JS Runtime In the Browser - JS engine + web Api's + callBack Queue + event loop

Web API's - are the functionalitiles provided ot the engine,accessible on window object

these api's are not part of the JS engine but the JS engine gets access to these

api's through global window object

example of web api - DOM , Fetch APi , timers ..

callBack functions - these are the DOM event handlers which are added in the code

example - onClick , timer, data...

EVENT LOOP - it takes the call back functions and put them in call stack so these function are executed

Execution Context in JS - it is an Environment in which javascript is executed .

Stores all the necessary information for some code to be executed.

Exactly one global execution context - Default context, created for code that is not inside any function

one execution context per function - for each function call, a new ec is created

compilation -> execution

inside execution -> creation of global execution context -> execution of top level code (inside Global EC) -> waiting for callbacks

EC contains all variable environment(let,const, var declaration , function, argument object), Scope chain, this keyword

these multi EC gets stacked in the call stack

SCOPE CONCEPTS - these are three types

Global | Function | Block (ES6)

- Outside of any function | - Variables are | - Variables are accessible only inside block

or block |accessible only inside | - however, this only applied to let and const (var is in function scope)

- Variables declared in |function, Not outside | - functions are also

global scope are accessible| - also called local | block scoped(only in strict mode)

everywhere | scope |

here Second scope can access first scope and these both can access global scope

but second scope cannot access second scope

HOISTING - a built-in language feature that moves variable, function,

and class declarations to the top of their scope before the code is executed

HOISTED? INTIAL VALUE SCOPE

function declaration yes Actual function Block

var variable yes undefined Function

let and const No <uninitialized>,TDZ Block

Function expression and arrow - depends if using var, let , const

TEMPORAL DEADZONE (TDZ)- makes it easier to avoid and catch errors : accessing variables

before declaration is a bad practice and should be avoided

//Example of hoisting fail -

//at this point the numProducts is undefined and matches the false value =0 ;

//so the code is executed

if(!numProducts) deleteShoppingCart();

var numProducts = 10;

function deleteShoppingCart(){

console.log('Deleted all products');

o/p will be - Deleted all products , even thou we have 1o products in numProducts

this shows the hoisitng disadvantage using var

BEST PRACTICES -

first step, just don't use var to declare variables.

Use const most of the time to declare variables and let, if you really need to change

the variable later.

Also in order to write clean code, you should declare your variables at the top of each scope.

That will just make your code at least look a little bit better.

Finally, always declare all your functions first and use them only after the declaration.

And this applies to all types of functions, even function declarations, which are hoisted.

So you could use function declarations before you declare them.

THIS KEYWORD - special variable that is created for every function.

takes the value of the owner of the function in which the this keyword is used

this is NOT static. It depends on how the function is called, and its value is only assigned

when the function is actually called.

method - this = <Object that is calling the method>

Simple function call - this = undefined (in strict mode)

ARROW function - this =<this of surrounding function >

event listener - this = <DOM element that the handler is attached to>

example -

const jonas = {

name : 'jonas',

year : 1999,

//calAge is method

//where this = jonas

calcAge: function(){

return 2025 - this.year; //this.year is better than using jonas.year

}

};

jonas.calAge();

o/p - 25

- this keyword always point to the object that is calling the method

PRIMITIVE vs OBJECT

example -

let lastName = 'Will';

let oldName = lastName;

lastName = 'Deaf';

console.log(lastName, oldName);

const jessy ={

firstName: 'Jessica',

lastName : 'MadMax',

age: 27,

};

const marriedJessy = jessy;

marriedJessy.lastName = 'Davis';

console.log('Before marriage:', jessy);

console.log('After marriage:', marriedJessy);

//o/p both the last name will be Davis

when we attempted to copy the original Jessica object,it did in fact not create a new object in the heap.

So, this one again, is not a new object in the heap.It's simply just another variable in the stack,

which holds the reference to the original object. So, both of these two variables simply point

to exactly the same memory address in the heap and that's because in the stack,

they both hold the same memory address reference.

//copying objects -

const jessy ={

firstName: 'Jessica',

lastName : 'MadMax',

age: 27,

};

jessyCopy = object.assign({}.jessy);

//object.assign will copy the object to and empty new object and then

//store it to the assigned variable jessycopy

//now when we change the variables of jessy copy it wont affect the original object

jessyCopy.lastName = 'Yeager';

console.log(jessy);

console.log(jessyCopy);

o/p: will be difference in the last name

Destructuring Array - extracting from the array

example - const restaurant ={

name: 'Ladieswaharum',

location: 'Italy',

categories:['Italian', 'Non-veg', 'VEG'],

starterMenu: ['Bread' , 'Kabab', 'Salad'],

mainMenu: ['Pizza','Pasta','Risotto'];

openingHours:{

thu:{

open: 12,

close: 22,

},

fri:{

open: 11,

close: 22,

},

sat:{

open: 0, //opens 24 hr

close: 22,

},

order: function(starterIndex, mainIndex){

return [this.starterMenu[starterIndex], this.mainMenu[mainIndex]];

},

};

//taking element out of array

const[first, ,second] =restaurant.categories;

console.log(first, second);

o/p will be Italian and VEG as we have give space in between it will skip the

second element of the array

//Switching variable

let [main , , secondary] = restaurant.categories;

console.log(main, secondary);

//normal way - const temp = main;

main = secondary ;

secondary = temp;

console.log(main ,secondary

// using destructuring

[main, secondary] = [secondary, main] ;

console.log(main , second);

//recieve two return value from a function

const [starter, mainCourse] = restaurant.order(2, 0);

console.log(starter, mainCourse);

const nested = [2 , 4 , [5,6]];

const[i, ,j] = nested;

console.log(i, j); //o/p will be 2 and nested array

const[i, ,[j,k]] = nested;

console.log(i, j , k); //o/p will be 2 , 5, 6

//destructuring an object

const { name, openingHours, categories} = restaurant;

console.log(name, openingHours, categories);

//what if we wanted variable name different from properties name

//this is helpful with dealing with 3rd party api data

const { name: restaurantName, openingHours:hours, categories:tags} = restaurant;

console.log(restaurantName, hours, tags);

//setting a default value to the variable

const{menu = [defaultValue], starterMenu:starter = [defaultValue] } = restaurant;

console.log( menu, starter);

//Mutating variables

let a =111;

let b = 999;

const obj = {a: 23, b: 7, c:14};

({a,b} = obj);

console.log(a,b);

//nested object

const{fri:{open, close},}=openingHours;

console.log(open, close);

SPREAD OPERATOR - takes the element out of the array and writes them manually

const arr = [7, 8, 9];

const badArr = [1, 2, arr[0], arr[1], arr[2]];

console.log(badNewArr);

o/p - [1,2,7,8,9]

//similarly we can use spread operator

const newArr = [1,2, ...arr];

console.log(newArr);

o/p - [1,2,7,8,9]

console.log(...newArr);

o/p - 1 2 7 8 9

const newMenu = [ ...restaurant.mainMenu, 'Avacado'];

console.log(newMenu);

o/p - ['Pizza','Pasta','Risotto', 'Avacado']

//copy array

const mainMenuCopy = [...restaurant.mainMenu];

//join 2 arrays

const menu = [...restaurant.starterMenu, ...restaurant.mainMenu];

console.log(menu);

o/p - complete menu array

REST PARAMETERS AND PARAMETERS -

//spread is used on the right side

const arr = [1, 2, ..[3,4]];

//Rest, bcuz on the left side

const [a, b, ...others] = [1,2,3,4,5];

console.log(a,b, others);

o/p - 1 2 [3,4,5]

//strings

const [pizza, ,risotto, ...otherFood] = [...restaurant.mainMenu, ...restaurant.starterMenu,];

console.log(pizza, risotto, otherFood);

o/p - Pizza Risotto and rest of the menu (pasta is skipped)

//objects

const { sat, ...weekdays} = restaurant.openingHours;

console.log(weekdays);

FOR - of LOOP -

const menu = [...restaurant.startMenu, ...restaurant.mainMenu];

for(const item of menu) console.log(item)

for(const item[i, el] of menu.entires()){

console.log(`${i+ 1): ${el}`);

}

ENHANCED OBJECT IN ES6 - helps us compute property name as well

const weekdays = ['mon' , 'tue' ,'wed','thu','fri','sat','sun'];

const openingHours = {

//instead of using the same old way to write we can change it to [with an expression]

[weekdays[3]:{

open: 12,

close: 22,

},

[weekdays[4]:{

open: 12,

close: 22,

},

[`day-$(2+4}`]:{

open: 0,

close: 22,

},

};

o/p -

day 6:{ }

fri: { }

OPTIONAL CHAINING - Es 2020 introduced optional chaining

for deeply nested objects with lots of optional properties

in optional chaining if a certain propterty doesn’t exist it will return undefined

//without optional chaining -

if(restaurant.openingHours && restaurant.openingHours.mon)

console.log(restaurant.openingHours.mon.open);

//with optional chaining

console.log(restaurant.openingHours.mon?.open);

.mon?.open - only if mon property exist in restaurant.openingHours object

only then the next property after question mark will be read

we can do the same with openingHours

console.log(restaurant.openingHours?.mon?.open);

LOOPING Objects : object keys , value, entries

const properties = Object.keys(openingHours);

console.log(properties);

SETS AND MAPS -

sets is collection of unquie properties it cannot have duplicate properties

const orderSet = new Set(['pasta', 'pizza', 'zaza' , 'rissoto' , 'pizza',]);

console.log(orderSet);

o/p - set(3) {"pasta", "pizza", "zaza", "rissoto"}

all the duplicate are gone, set looks similar to array and just like array , SET is also

iterables , order of element in set is irrelevant and the elements are unquie

console.log(orderSet.size); // gives the size of the SET

console.log(orderSet.has('pizza')); //check for the element

orderSet.add('Garlic Bread'); //adds the element but only unquie value

orderSet.delete('Garlic Bread'); //delete from set

orderSet.clear(); //clears the set

there is no need to get data out of set bcuz every value is unquie and order dosent matter

then there is no point in retriving the value from the SET we can check with the HAS method

looping is possible in the set

main use case is to remove the duplicate values from an array

example :

const staff = ['Waiter', 'Chef', 'Waiter','Manager', 'Chef', 'Waiter'];

const staffUnique = new Set(staff);

console.log(staffUnique);

o/p - shows the set

console.log(new Set(['Waiter', 'Chef', 'Waiter','Manager', 'Chef', 'Waiter']).size);

MAPs - Maps are more useful than SETS

Maps datastructure which we can use to map values to keys

just like objects data stored in key value pair

the main difference between objects and maps is that the keys can be of any type

unlike in objects which has string as keys

const rest = new Map(); //creates an empty map

rest.set('name','Classico Italiano');

rest.set(1, 'Firenze, Italy');

console.log(rest.set(2, 'Lisbon, portugal'));

rest.set('categories', ["pasta", "pizza", "zaza", "rissoto"])

.set('open', 11)

.set('close', 23)

.set(true, 'We are open :D')

.set(false,'We are close :(');

console.log(rest.get('name'));

console.log(rest.get(true));

// while retriving the datatype of the key matters (rest of the code in IDE

map iteration -

//convert object to map

console.log(Object.entries(openingHours));

const hoursMap = new Map(Object.entries(openingHours));

console.log(hoursMap);

WHICH DATA Structure to Choose -

there are essentially 3 sources of data -

1)from the program itself

2)from the UI - data input from the user or data written in DOM

3)from external soruces - web API

if its a simple list - use Arrays or sets

if its a key value pair - objects or maps

Arrays vs Sets

- use when you need ordered -Use when you need to work with

list of values(might contain Unique values

duplicates) - Use When high performance is really

- use when you need to importance

manipulate data - use to remove duplicates from arrays

Objects vs Maps

-More 'traditional' key/value -Better performance

store - keys can have any datatype

- Easier to write and access - Easy to iterate and compute size

- use when you need to include - use when you simply need to map key to values

functions(methods) - use when you need keys that are not strings

- use when working with JSON

STRINGS -

const airline = 'AIR INDIA';

const plane = 'A320';

//getting the element at a positon

console.log(plane[0]);

or

console.log('A320'[0]);

//getting the length of the string

console.log(airline.length);

//positon in which the certain letter is

console.log(airline.indexOf('r));

console.log(airline.lastIndexOf('r'));

console.log(airline.indexOf('INDIA')); //this is case sensitive defaultoutput will be -1

//slice - position at which it starts to attract

console.log(airline.slice(4)); //returns a substring as strings are immutable we will have to store it in a variable to use it

console.log(4, 6));//starting index and ending index

//without hardcoding

console.log(airline.slice(0, airline.indexOf(' ')));

console.log(airline.slice(airline.lastIndexOf(' ')+1));// +1 to eleminate space

when we call a method to a string javascript will convert that string primitive to a string object

with the same content and then its on that object where the methods are called this is called BOXING

bcuz it takes our string and puts it into a box which is an object

Resume

Charter Spectrum - Charter is a well known mobile seller, Major market is present in America.

Model View Controller (MVC)

As an Analyst and Solution Provider, primary responsibility was to meticulously analyze Jira tickets related to software bugs/errors,

prioritizing them based on severity and impact. identifying, understanding, and resolving bugs within the Spring MVC and Java technology stack.

Additionally, utilizing Splunk logs for effective troubleshooting and debugging.

Frontend Developer (React)

Technologies: React, CSS, Jest, Git, etc.

Developed and designed responsive UI components using React and CSS to deliver a seamless user experience.

Created reusable and modular components that were utilized by multiple teams to ensure consistency and efficiency across projects.

Worked on debugging, troubleshooting, and resolving bugs to enhance the functionality and performance of applications.

Wrote unit tests using Jest to ensure the reliability and stability of React components.

Collaborated closely with UX/UI designers and backend teams to integrate APIs and improve overall product functionality.

Followed best practices for code quality, version control, and deployment workflows.

POS system