**-------------------------------------------------------------------JS----------------------------------------------------------------**

**The statement to print any thing =>** console.log("This is for printing statement"); // : terminator is imp always

**DATATYPES IN JS:**

var(can be used globally in entire code ), let(can be used in a block where it is defined only ), const(can be used in block it is defined and cant change its value)

**operators are as same as C and CPP assignment , logical , arithmetic , unary and etc.**

**if statements are:**

> If

> IF else

>IF else if

>terniary conditional statement : condition ? statement1 : statement2;

**loops:**

***>for loop:***

for(let i=1;i<=5;i++){

statement----

}

*>****while loop:***

while(i<5){

statement---

i++;

}

>do while loop:

do{

statement---

i++;

}while(i<5)

***>for-in loop:***

for(let key in objVar){

statements---

}

***>for-of loop:***

for(let val of strVar){

statement----

}

**string:**

**(NOTE: Strings are immutable means it can't be changed once its defined.)**

**>create string:**

let str="Shivansh";

**>string length:** str.len;

**>string indices:** str[0],str[1],str[2] // to access the particular letter at a particular index

**>Template literals**:

let result = `your marks is ${student.mark}`;

console.log(result);

**>STRING METHODS:**

->let str1="Shiv";

***-> str.UPPERCASE();***

***-> str.LOWERCASE();***

***-> str.trim(); //removes the white spaces from the string***

-> console.log(str1.slice(1,4)); // str.slice(start,end) it will print the string from start to end-1

-> let str2="Ansh";

let res= str1.concat(str2); // str.concat(str2) it will concatenate the two strings means it will add both strings

console.log(res);

->newRes= "My name is " + str1 + str2; // we can also concatenate the string using + operator

console.log(newRes);

->console.log(str1.replace("h" ,"k")); // str.replace("old value","new value") it will replace the old value with new value

->console.log(str1.charAt(2)); // str.charAt(index) it will return the character at the given index

**ARRAY:**

**(Note: Arrays are mutable means it can be changed )**

-> A variable which stores multiple values

-> can access the element of array by index numbers

-> Array method:

> push(): ADD to end

> pop(): DELETE fromm end & return

> toString(): Convert array to string

> concat(): joins multiple array & return result

> unshift(): add to start

> shift(): delete from start & return

>slice(): returns a piece of array:

slice(startidx,endidx);

>splice(): change original array(add,remove,replace):

splice(startidx,delCount,newE(1....));

**Function:**

=> define the function:

function funtionName(){

//statements

}

=> function Call:

functionName();

=>Function calling by parameters:

function FunctionName(x,y){

//Statements

}

functionName(1,2);

=> ARROW FUNCTION:

const arrowFunction=(x,y)=>{

statements;

}

=>Array method of functions:

=>forEach loop function:(we can pass value,index and array it self by using this function)

arr.forEach((val)=>{

console.log(val);

})

>forEach is higher order function.

(NOTE: Higher order functions are the function which can return function or using another function as a parameter)

=>MAP: MAP creates new array with the result of some operation. The value its callback returns are used to form new array.

syntax- arr.map(callbackFnx(value,index,array))

ex.let nums = [45,43,24,,47];

nums.map((val) =>{

console.log(val);

})

=> Filter: Creates a new array of elements that gives true for a condition/filter.

eg. all even elements

arr=[1,2,3,4,5,6,7];

let newArr=arr.filter((val)=>{

return val % 2 ===0;

});

console.log(newArr);

=> Raduce method: perform some operation & reduces the array to a single value. It returns that single value.

eg.

let arr=[1,2,3,4];

const output = arr.reduce((prev,curr)=>{

return prev > curr ? prev:curr ;

});

console.log(output);

here in this code compiler takes the first element of array as prev and next element as curr and performs the operation mentioned in the retun statement it can be any thing like add ,subtract ,mul,divide.

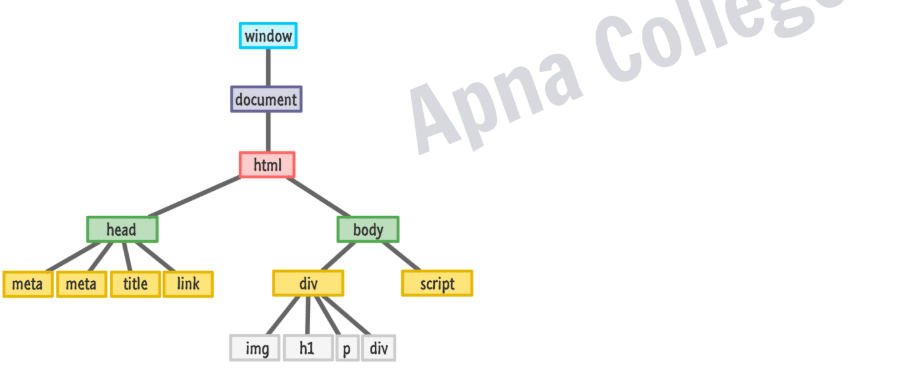
In this case we are prefroming which is the greater one in the array so first prev is 1 and curr is 2 compares between them 2 is our new perv and 3 is the new curr comparing them 3 is the net prev and 4 in curr if prev > curr the print perv else curr as the return statement says that our answer is 4 on the place of curr.

**------------------------------------YOU HAVE COMPLETED THE BASICS OF THE JS------------------------------------**

**==================================DOM===========================================**

**=> DOM: When a web page is loaded, the browser creates a DOCUMENT OBJECT MODEL (DOM) of the page.**

**this is stored in window object when a webpage is loaded.**

****

**=>DOM manipulation:**

**> Selecting with id:**

**syntax- document.getElementById("myid");**

**>Selecting with class:**

**syntax- document.getElementByClassName("myClass");**

**>Selecting with tag:**

**syntax- document.getElementByTagName("p")**

**>document.querySelector(“#myId / .myClass / tag”)**

**//returns first element**

**Query Selector**

**document.querySelectorAll(“#myId / .myClass / tag”)**

**//returns a NodeList**

**>property you should remember:-**

**-tagName: returns tag for element nodes.**

**-innerText: returns the text content of the element and all its children.**

**-innerHTML: returns the plain text or HTML contents in the element.**

**-textContent: returns textual content even for hidden elements.**

* **DOM Attribute:**
* **getAttribute(attr) //to get the attribute value**
* **setAttribute(attr,value) //to set the attribute value**
* **Style**
* **node.style**
* **Insert Elements:**
* **node.append(el) // adds at the end of node (inside).**
* **node.prepend(el) //adds at the start of node (inside).**
* **node.before(el) //adds before the node(outside).**
* **node.after(el) //adds after the node (outside).**
* **Delete Elements:**
* **node.remove() //remove the node.**
* let newBtn = document.createElement("button");//creating a variable and storing a element buton in it
* newBtn.innerText = "Click me!"; //inserting text in it
* newBtn.style.color="white"; //giving the text color
* newBtn.style.backgroundColor="red"; //giving button background color
* document.querySelector("body").prepend(newBtn); //document.querySelector("body") here works as calling the body tag elements and prepend newBtn is adding the element stored in the newBtn before the body tag
* **Events in JS**
* **Events are fired to notify code of "interesting changes" that may affect code execution.**
* **The change in the state of an object is known as an Event**
* **Mouse events (click, double click etc.)**
* **Keyboard events (keypress, keyup, keydown)**
* **Form events (submit etc.)**
* **Print event & many more > https://developer.mozilla.org/en-US/docs/Learn\_web\_development/Core/Scripting/Events**
* **Event Handling in JS:**

**node.event = ( ) => {**

**//handle here**

**}**

* **Event Object**
* **It is a special object that has details about the event.**
* **All event handlers have access to the Event Object's properties and methods:**

**node.event = (e) => {**

**//handle here**

**}**

**e.target, e.type, e.clientX, e.clientY**

* **Event Listeners**
* **node.addEventListener( event, callback )**
* **node.removeEventListener( event, callback )**

**\*Note : the callback reference should be same to remove**

* **CLASSES AND OBJECTS:-**
* **Prototypes in JS:-**

**A javaScript object is an entity having state and behavior (properties and method).**

**JS objects have a special property called prototype.**

**We can set prototype using \_ \_ proto \_ \_**

**\*If object & prototype have same method,**

**object’s method will be used.**

* **CLASSES IS JS:**

**Class is a program-code template for creating object.**

**Those object will have some state (variable) & some behaviour (functions) inside it.**

**Syntax:**

**class MyClass{**

**constructor(){..}**

**myMethod(){…..}**

**}**

**Let myObj = MyClass();**

* **Constructor() method in js:**
  + - * **Automatically invoked (executed) by new.**
      * **Initialize object.**
* **Inheritance in JS**
* **inheritance is passing down properties & methods from parent class to child class.**
* **\*If Child & Parent have same method, child’s method will be used. [Method Overriding]**

**Syntax:**

**class Parent {…..}**

**class Child extends Parent {…..}**

* **Super Keyword:**

1. **The super keyword is used to call the constructor of its parent class to access the parent's**
2. **properties and methods.**
3. **super.parentMethod( args )**

**Sync in JS**

**Synchronous means the code runs in a particular sequence of instructions given in the program.**

**Each instruction waits for the previous instruction to complete its execution.**

**Due to synchronous programming, sometimes imp instructions get**

**blocked due to some previous instructions, which causes a delay in the UI.**

**Asynchronous code execution allows to execute next instructions**

**immediately and doesn't block the flow.**

**Synchronous**

**Asynchronous**

**Callbacks**

**A callback is a function passed as an argument to another function.**

**Callback Hell**

**Callback Hell : Nested callbacks stacked below one another forming a pyramid structure.**

**(Pyramid of Doom)**

**This style of programming becomes difficult to understand & manage.**

**Promises**

**Promise is for “eventual” completion of task. It is an object in JS.**

**It is a solution to callback hell.**

**let promise = new Promise( (resolve, reject) => { .... } )**

**Function with 2 handlers**

**\*resolve & reject are callbacks provided by JS**

**Promises**

**A JavaScript Promise object can be:**

**Pending : the result is undefined**

**Resolved : the result is a value (fulfilled)**

**Rejected : the result is an error object**

**resolve( result )**

**reject( error )**

**\*Promise has state (pending, fulfilled) & some**

**result (result for resolve & error for reject).**

**Promises**

**.then( ) & .catch( )**

**promise.then( ( res ) => { .... } )**

**promise.catch( ( err ) ) => { .... } )**

**Async-Await**

**async function always returns a promise.**

**async function myFunc( ) { .... }**

**await pauses the execution of its surrounding async function until the promise is settled.**

**IIFE : Immediately Invoked Function Expression**

**IIFE is a function that is called immediately as soon as it is defined.**