1. **“Java Script Hello world program”**

Open visual studio code – File – New File – Save(ctrl+s) give the file name extension ( .js ).

When you create a new file you need to give .js extension that JavaScript related syntax

will get highlighted in your editor.

print something in output using JavaScript - console.log("Hello World")

So how to run this program? ur at this path where this file is present in users owner directory.

Basically node is an engine which will help you to execute your JavaScript based files.

node basics.js – If you give this in terminal it will run the program

// these are comments. These will not execute and it will not impact test execution.

/\*

if you have comments in multiple lines

\*/

1. **Declaring the variables in javascript**
2. **Understanding the datatypes in Javascript**

variables can hold values with any type of data.

in JavaScript there are different engines. – ES5,ES6,ES7,ES8 - These are nothing but different versions.

until Ees5 version for storing our any variable value, we use a keyword called Var, but from ES6 engine version, they have introduced two more keywords to represent variable that is let and const.

how you store a value in variable- let a=4

to print the varaiable value - console.log(a)

what kind of datatype it is – typeof() is a method this method will tell you the datatype

console.log(typeof(a))

And for every variable, we are using this let keyword so that it recognizes this as a variable.

let b = 234.6

console.log(typeof(b))

let c = "Hari Sankar"

console.log(typeof(c))

let required = true

console.log(typeof(required))

// two more other data types - null & undefined

if you store null value into any variable, then that data type is null and undefined

is like if you don't create and assign any value to the variable, then that returns undefined data.

can I use any assignment operators on these values?

So there is one negation operator called not ( ! ) apart from this arithmetic operations.

var c = "Hari Sankar"

console.log(typeof(c))

let required = true

console.log(typeof(required))

var c = a+b

console.log(c)   // we cannot redeclare with let keyword but possible with var

console.log(!required)    // if required is true if you add ! opertor then it will turns false

What do you mean by Reassigning ?

let c = "Hari Sankar"

console.log(typeof(c))

let required = true

console.log(typeof(required))

// null & undefined

//let c =a+b( it did not work // we cannot redeclare with let keyword but possible with var)

 c = a+b  // reassigning is allowed with let

 //var  c=a+b (this is also allowed)

console.log(c)

When you assign a value to const, that should stick constant in your entire script to duration.

So you can use this const keyword in your program whenever you think that this variable cannot be changed in your program. ( Ex : const c = “Hari Sankar” )

if you think that it can be reassigned to another value, then go with the let.

if you think that this variable should be redeclare that again, another time in your program,

then go with var.

console.log("Hello World")

let a=4

console.log(a)

console.log(typeof(a))

let b = 234.6

console.log(typeof(b))

let c = "Hari Sankar"

console.log(typeof(c))

let required = true

console.log(typeof(required))

// null & undefined

//let c =a+b( it did not work // we cannot redeclare with let keyword but possible with var)

 c = a+b  // reassigning is allowed with let

 //var  c=a+b (this is also allowed)

console.log(c)

console.log(!required)    // if required is true if you add ! opertor then it will turns false

1. **Loops and conditions in javaScript**
2. **Logical operators and assignments in js**

Create new file – save the file with basics2.js

//if that expression is true, then it will go inside this block.

//The expression we are converting into false, but this variable value will still be true.

const flag = true

if(!flag)

{

    console.log("condition satisfied")

}

else

{

    console.log(flag)

    console.log("condition not satisfied")

}

/\*While loop - if the expression is true, this loop will keep on executing until this condition.

  become false. If this never become false, then this loop will go to infinite. \*/

let i=0

while(i<10)

{

    i++

    console.log(i)

}

let i=0

while(i>10)

{

    i++

    console.log(i)

}

/\*Do while loop -  no matter if condition is true or false.

One round of loop will definitely run and then only it will check for condition.

when you write a do loop, you have to end your  loop with semicolon \*/

do

 {

 i++

} while(i>10);

console.log(i)

when we already have while loop, then why did we choose for loop?

 basically What I have achieved with while loop same thing I'm trying to do with for loop.

So all these three actions we are wrapping into one single for block. And then we are going inside.

When I have to use while loop ? when I have to use for loop ?

for loop that how many times you have to run this loop.

In while loop it’s not just about run You can use it for any statement also because while only looks if the condition is true.

/\*how many times you have to repeat loop Then you can go for a for loop

But if you want to repeat loop based upon some condition evaluation that

if it becomes true or false,then choose while loop\*/

for(let k=0;k<=10;k++)

{

    console.log(k)

}

let required

while(required)

{

    console.log(required)

   required = false

}

So given two values and you want to print the numbers which can be commonly retrieved from 1 to 10.

When you use break it will break the statement.

// from 1 to 10 give me common multiple values of 2 and 5

console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

//let  n = 0;

for(let k=1;k<=10;k++)

{

    if(k%2 ==0 || k%5 ==0)       // && if both should should be satified then only it go inside the loop

    {                          // if you use or operator || then if and one is satified it go inside the loop

       // n++

        console.log(k)

       // if(n ==3)

        //break

    }

  }

1. **Arrays and its methods with detailed examples**
2. **Array functions**

Create new file and save it as basics2.js

Array is a collection of elements.

So if you have a set of values, then you can store all them in one container called array.

So this array exposes three powerful functions(reduce,filters,map) to reduce to core complexity.

when to use the filter method when to use the reduce method ?

So if you have any concept where you want to iterate and update values like accumulating like this. So he had the sum is accumulating with new value every time you iterate, if you have such scenarios, then use reduce method.

But if you have a scenario where you want to filter, your array based upon some conditions like even numbers or odd numbers, then use filter method and give that condition here.

var marks = Array(6) // subject

var marks = new Array(20,40,35,12,37,100)  // subject marks

            //or

var marks = [20,40,35,12,37,100]

console.log(marks[2])   //  2 is index value

marks[3]=14  //14 number is assigned to the 3rd index when u print the 3rd index you will get 14

console.log(marks)//[20,40,35,14,37,100]

console.log(marks.length)//6

marks.push(65)   // add new element at the end of an array

console.log(marks)//[20,40,35,14,37,100,65]

marks.pop()//[20,40,35,14,37,100] // pop remove the last element of array

marks.unshift(12)   // Insert new element at the start of an array

console.log(marks) //[12,20,40,35,14,37,100]

//create sub array from the main array.

var marks = new Array(20,40,35,12,37,100)

submarks = marks.slice(2,5)

console.log(submarks)

console.log(marks.indexOf(100)) // if you want to know know the index no based on value

console.log(marks.includes(120))  // Includes is a methods to cehck the value is present or not in the array.

//print all the elements present in this array

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

{

    console.log(marks[i])

}

//sum of all the elements present in the array

var sum = 0

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

{

    sum = sum + marks[i]

}

console.log(sum)

// array exposes three powerful functions(reduce,filters,map) to reduce to core complexity.

//reduce method Can be used to perform operation on all the elements

/\*reduce method - every iteration it will pull each and every value from an array and

  it will place that value in this mark variable for every iteration.\*/

let total = marks.reduce((sum,mark)=>sum+mark,0)   // we are done sum here instead of for loop we use reduce method

console.log(total)

/\* old array [12,13,14,16] create new array with even numbers of scores array [12,14,16]

and multiply each value with 3 and sum the array [12,14,16] \*/

 var scores = [12,13,14,16]

 // create new array with even numbers of scores

var evenScores = []

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

{

    if(scores[i] % 2 == 0)

    {

        evenScores.push(scores[i])

    }

}

console.log(evenScores)

    // or use filter method

    let newfilterevenscores = scores.filter(score=>score%2==0)

    console.log(newfilterevenscores) // [ 12, 14, 16 ] mulitple by 3 [36,42,48]

// map function will modify each and every value of array to new value.

let mappedArray = newfilterevenscores.map(score=>score\*3)

console.log(mappedArray)  // [36,42,48]

// sum

let totalVal = mappedArray.reduce((sum,val)=>sum+val,0)

console.log(totalVal)   // 36+42+48 = 126

                   //or

var scores1 = [12,13,14,16]

let sumvalue = scores1.filter(score=>score%2==0).map(score=>score\*3).reduce((sum,val)=>sum+val,0)

console.log(sumvalue)

/\*How can we perform sorting on any array

2 types of sorts - sorting an array which have strings and sorting an array which have numbers\*/

let fruits = ["banana","mango","pomegrante","apple"]

console.log(fruits.sort())

console.log(fruits.reverse())

var scores2 = [12,003,19,16,14]

//console.log(scores2.sort())

//scores2.sort(funtions(a,b){

    //return a-b

//})

console.log(scores2.sort((a,b)=>a-b))

1. **what are the functions and how we declare functions and what are their usage in JavaScript.**
2. **Major difference b/w var,let,const**

this difference is related to the scope where they can be declared and from where they can be accessed.

the scope of this var keyword is global. If it is declared in the global level

the scope of this var is function only . If it is declared in the function level.

if you declare the Var keyword inside the function, then you cannot access it outside.

when you define var keyword in function level, the scope will die in this function only.

//a block of code can be executed together by wrapping them in a module called function.

function add(a,b)

{

    return a+b

}

let sum = add(2,3)

console.log(sum)

// do not have name => Anyonyms function ---- Function expressions

let sumofIntegers = function(c,d)

{

    return c+d

}

       //or

let SumOfNumbers = (c,d) => c+d

console.log(SumOfNumbers(2,3))

//var scope - global/functional

//let - global/block level { }

let greet = "Evening"  //global

if(1==1)

{

    let greet = "Afternoon"

}

function add(a,b)

{

    let greet = "Morning"

    return a+b

}

console.log(greet)

1. **Strings & its methods in Javascript**

//strings and its methods in JavaScript.

let day = 'tuesday '    // string

console.log(day.length)   //8

let subDay = day.slice(0,4)

console.log(subDay)

console.log(day[1])  // 1 is the idex no //u

//tue   day

let splitDay = day.split("s")

console.log(splitDay[1].length)

console.log(splitDay[1].trim().length)

//parseInt -  is string to covert numbers

let date = '23'

let nextDate ='27'

let diff = parseInt(nextDate) - parseInt(date)

console.log(diff)

diff.toString()  // toString - convert number to string

//how to concatenate two strings

// "+" operator concatenate two different strings

let newQuote = day+ "is Funday"

console.log(newQuote)  //  tuesday is Funday

let val = newQuote.indexOf("day")   //Index of day word starts

console.log(val)  //4

//if you want to the second "day word" of index

let val1 = newQuote.indexOf("day",5)   //Index of day word starts

console.log(val1)  //14

//tuesday is Funday  - How many day word is present

let count = 0

let value = newQuote.indexOf("day")

while(value!== -1)

{

    count++

value = newQuote.indexOf("day",value+1)   // value+1 - 1st day identifed by 4th index

                                    //so value =4. 4+1=5 search starts from 5th index

}

console.log(count)  // 2

1. **What are Javascript object ?**

/\*JavaScript objects - object is a collection of properties.

if you think that any object(person) hold multiple properties(key-string, value- any datatype pair)

so you can wrap up all these details in one single object and you can just call that property with object dot property.\*/

let person = {

    firstName : 'Tim' ,

    lastName  : 'joe' ,

          age : 24   ,

          fullName : function()

          {

            console.log(this.firstName+this.lastName)  //concatenate

          }

}

console.log(person.fullName())  // Timjoe

console.log(person.lastName)  // dot notation

console.log(person['lastName'])  // array like notation

person.firstName = 'Tim Dane'    // Updated in the existing properties

console.log(person.firstName)

person.gender ='male'  // Added in the properties

console.log(person)

delete person.gender

console.log(person)

console.log('gender' in person)//check if the property exist in the object.

//if you want to print the values of all the properties

for(let key in person)

{

    console.log(person[key])

}

1. **Classes and its Properties, Methods in JavaScript ?**

**Imprting & exporting and inheritance classes**

/\*Classes and its Properties, Methods in JavaScript ?

classes, which is an object oriented principle in JavaScript.

So these classes are introduced only from ES6 engine.

two different ways you can define properties in class

Constructor is a method which executes by default when you create object of the class

constructor will be default called when you create object\*/

module.exports = class person

{

    age = 25

    // location = "canada"

    get location()

    {

        return  "canada"

    }

    constructor(firstName,lastName)  // Instance variable

    {

             this.firstName = firstName

             this.lastName = lastName

    }

    //methods

    fullName()

    {

        console.log(this.firstName,this.lastName)

    }

}

// let person0 = new person0("Tim", "joseph")

// let person1 = new person1("Oris", "jones") // create object of the class

// console.log(person.age)

// console.log(person0.location)

// console.log(person0.fullName())

// console.log(person1.fullName())

/\*Inheritance is the multiple pillar in object oriented programming

one class can inherit/acquire the properties,methods of another class

The class which inherits the properties of other is known as subclass (derived class,child class)

the class whose properties are inherited is known as superclass\*/

// rule 1 - if parent class have any constructor, then your child to go to class also should implement same kind of constructor.

//rule 2 – call the parent class constructor in the child class constructor

// super keyword W'll just go ahead and call parent class constructor.

const person = require("./basics7")

class pet1 extends person

{

    get location()

    {

        return "BlueCross"

    }

    constructor(firstName,lastName)

    {

        super(firstName,lastname)

    }

}

let pet1 = new pet1("sam","sam")

pet.fullName()

console.log(pet1.location)