**Variables in JavaScript:**

JavaScript is a dynamically typed language which means you can change the variable during runtime. variable is a container in JS that stores the value, the value of the variables in JS can be changed during runtime.

Var a = 7 – here **a** is called identifier, and 7 is called literal and this process is called Declaring variable.

**Rules for choosing a variable**: 1)Letters, Digits, Underscore and $ sign allowed

2) Can not start with a number and follow case sensitivity

3) The reserved keywords in JS can not be used as a variable

**Let, var and Const :**

Var is globally scoped where let and const are block scoped. Block meaning inside a curly braces.

Var can be updated and re-declared within its scope but let can be updated but could not be redeclared. Const can neither be updated and re-declared. We will use const only when we know that the value of the variable will not change for the entire program. Var variables are initialised with undefined where let and const are not. Const must be initialised during declaration unline let and var.

Re-declared: Let a =5

Let a = 7

This is called redeclaring.

Update: Let a = 5

a = 7

This is called updating the value of the variable.

**Primitive data types and Object in JS:**

There are 7 primitive datatypes in JS. Non-primitive data types are called object.

7 primitive data types: Null, Number, Symbol, String, Boolean, Bigint, Undefined

To find datatype we can do, console.log(typeof a)

**Objcets:** Object in Js are key and value pair. Key meaning data type-

EG: const item = {

“harry” = true,

“Ram” = False,

“Subh” = 35

}

We can do look up of object and do mapping using key-value pairs.

**Practice set question- Basic:**

1. Create a variable of type string and try to add a number to it. (Concatenation concept)
2. Use type of function to find out the datatype of a variable.
3. Create a const object in Js and can you change it to hold a number later- No, it can not be done.
4. Try to add a new key to the const object in problem 3, were you able to do it ?
5. WAP to create a word meaning dictionary of 5 words.

**JavaScript Operators and Expressions:**

A fragment of code that produces a value called an expression. Every value written in JS is literally an expression. Suppose, Harry or “+”

**Operators**: Addition(+), Multiplication(\*), Division(/), Subtraction(-), Exponentials(\*\*), incremental(++), decrement(--), modulus(%)

**Comparison Operators:** equal to(==), not equal(!=), equal value and type of the variable both(===), not equal value and type of the variable both(!==), greater than(>), less than(<). Greater equal(>=), less equal(<=)

**Logical Operators:** It generally works on Boolean, &&(logical and), logical OR (||), logical NOT(!)

**Conditional Expressions:**

Sometimes, we might have to execute a block of code based on some condition. In Js we have three types of if-else statements.

1. If 2) If-else 3) if-else if -else statement

**Prompt() and alert() function in JS:**

prompt() instructs the browser to display a dialog with an optional message prompting the user to input some text, and to wait until the user either submits the text or cancels the dialog.

Let a= prompt(“what is your age”)

Alert(): It display a message.

**Typecasting string into number**- Number.ParseInt(a)

1)IF : It enables the condition inside a bracket if the condition is evaluated true then, the code executes, otherwise it goes under the else bracket.

If the condition is true then it goes under if and if the condition is false then it goes under else.

**Ternary operator:**

*console.log*("you can do", (a>5 ? "bad" : "good"))

**Loop and Function in JS:**

For loop, While loop, Do-While loop, For-IN loop, Foor-Of loop

Usage: We use loop to perform a repeated action.

**Type of Loops:**

1. For Loop: Loop a block of code, no of times
2. For In Loop: Loop through the keys of an object
3. For Of Loop: Loop through the values of an object
4. While Loop: Loop a block based on a specific condition
5. Do-While Loop: While Loop variant which runs at least one.
6. For Loop: for(statement1; statement2; statement3)

Statement1-> it executes only one time

Statement2-> It is a condition, if condition is true then only loop body executes

Statement3-> It executes everytime when loop body is executed

**while and do-while loop:**

In while loop first the condition is checked then the block runs. But in do-while loop first the block runs after that the condition got checked. Do-while loop at least runs once before the condition got checked.

**Function in JS:**

Function is a block of code assigned to perform a particular task.

When we use a repeated logic again and again and use that in our code, for the betterment of code we can separate the logic and put that in a function so that from the next time onwards we just call it and it work.

Function invocation:- function(arg1,arg2) ;

Function invocation is a way to use the function inside a code.

**let a=parseInt(prompt("take first number"));**

**let b=parseInt(prompt("take second number"));**

**function sum(a,b){**

**return 1+((a+b)/2)**

**}**

**console.log("the value is", sum(a,b))**

**Arrow function:**

**c**onst hello=()=>{

Console.log()

}

**c**onst sum=(a,b)=>{

return a+b

}

Practice set:

Q1) create and access an object of marks and use for loop

const *marks*={

*Riya*:44,

*Simi*:67,

*Jinia*:8,

*Ammna*:5

}

//*create for loop*

*for*(leti=0; i<*Object.keys*(marks)*.*length; i++)

{

*console.log*("The marks of " + *Object.keys*(marks)[i] + "is" + marks[*Object.keys*(marks)[i]])

}

**String in JS:**

Strings are used to store and manipulate text. It is collection of character. It can be created using the following syntax-

Let name = “Deblina”

**Template Literal:**

It use backticks instead of quote to define a function. We can use $ sign to refer variable. With Template literal it is possible to use both single quoted and double quoted inside a string.

We can insert variable directly in Template Literal, this is call **string interpolation**. **${variable}**

leta1="Deblina"

leta2="Indranil"

letsentence=`${a1} *is a friend of* ${a2}`

*console.log*(sentence)

**Escape sequence character:**

We can use single quote escape sequence character inside our string.

Let fruit = r-carriage return, t-tab, n-new line

letfruit='App\'le'

*console.log*(fruit)

letfruit1='App\nle'

*console.log*(fruit1)

letfruit2='App\tle'

*console.log*(fruit2)

**Methods in String:**

String is immutable, it can never be changed but new string can be made out of it. Difference between property and a function.

Name.length ->property

Name.toUpperCase() - > method

Name.toLowerCase() -> method

Escape sequence character consider as one charecter.(eg: back slash n)

Name.slice(2,4)-> print 2nd and 3rd position

Name.slice(2) ->print 2nd position to last

Name.replace(“Deb”,”Rim”) ->the deb position in string will be replaced by Rim

Name.concat(“I am a very nice”, Name, “ok done”)

Name.trim() -> it will remove all the blank space from string

**Use .includes in string**: It return **true** or **false** value

//*use .includes*

letsentence=`*Deblina is a very good girl*`

letword="very"

*console.log*(*sentence.includes*(word))

*console.log*(`the word ${word} ${*sentence.includes*(word) ? "is" : "is not"} there`)

**Array in Js:**

1. An array can hold many values under a single name, and you can access the values by referring to an index number.
2. Arrays are variable which can hold more than one value.
3. Syntax: let arr= square bracket
4. Array will return undefined if we try to access an index which doesn’t come in array list.
5. he value of array can also be change by using index number.
6. We use const variable to define an array mostly. But It does NOT define a constant array. It defines a constant reference to an array. Because of this, we can still change the elements of a constant array.
7. Arrays are typeof object.

**Methods in Array:**

toString() -> convert an array into a string of comma-separated value

join() -> join all the array elements using a separator

pop() -> remove the last element from the array and return the popped element

push() -> add new elements at the end of the array and return the length of the array

shift() -> removes the first array element and "shifts" all other elements to a lower index. The shift() method returns the value that was "shifted out"

unshift() -> the unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements. The unshift() method returns the new array length.

delete -> it is not a method but a operator. After deleting also the length of the array stays same. It doesn’t change.

letb= *cars.push*("Skoda")

*console.log*(cars, b)

delete cars[4]

*console.log*(cars)

*console.log*(cars[4])

(6) ['BMW', 'BENZ', 'TOYOTA', 'Audi', 'jaguar', 'Skoda'] 6

practice 401.js:14

(6) ['BMW', 'BENZ', 'TOYOTA', 'Audi', …, 'Skoda']

practice 401.js:16

undefined

So, it means the value just get deleted but the memory space still remains and it reutn a undefined value.

concat() ->use to join array from a given array-. It doesn’t change existing array but create anew array.

new\_arry = oldarray.concat(concatenated\_array)

sort() -> it sort an array alphabetically. It changes the original array.

**Numeric Sort:** By default, the sort() function sorts values as strings.

This works well for strings ("Apple" comes before "Banana"). However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".Because of this, the sort() method will produce incorrect result when sorting numbers.

You can fix this by providing a compare function:

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});

**The Compare Function:** the purpose of the compare function is to define an alternative sort order. The compare function should return a negative, zero, or positive value, depending on the arguments:

***function(a, b){return a - b}***

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

If the result is negative, a is sorted before b. If the result is positive, b is sorted before a. If the result is 0, no changes are done with the sort order of the two values.

**Example:** the compare function compares all the values in the array, two values at a time (a, b). When comparing 40 and 100, the sort() method calls the compare function(40, 100).

The function calculates 40 - 100 (a - b), and since the result is negative (-60),  the sort function will sort 40 as a value lower than 100.

***NOTE:*** *The values in the num array are all numbers, so when the sort() method compares them, it knows that it is dealing with numbers and performs the comparison accordingly. Therefore, there is no need to explicitly define the type of a and b in the sorting function*

**reverse()** - > reverse method reverses the elements in an array.

**Splice()** -> It can be used to add new items in array and also deleted existing items from the array depending on the index position

**Slice()** - > it will remove the items from an array and make a new array

**Loops with Array**

forEach Loop: *{*

*num.forEach*((parameter)=>

{*console.log*(parameter+1)})

It call s function for each array element. forEach method will only works on array. It takes one argument and do the required work on that.

forEach can take three arguments also. forEach(value,index,array)

**Array.from** -> use to create an array from any another object. Maybe when we want a string to change an array we will use this.

Array from string:

letname="Deblina Karmakar"

*console.log*(*Array.from*(name))

O/P: 16) ['D', 'e', 'b', 'l', 'i', 'n', 'a', ' ', 'K', 'a', 'r', 'm', 'a', 'k', 'a', 'r']

**Map, Filter and reduce in JS:**

This is higher order array methods.

**map()** -> creates a new array by performing some operation on each array element. We used map when we needed to make a new array. Return an array.

letarr=[1,3,9,4,6]

letp=*arr.map*((*element*,*index*)=>{

*console.log*(element,index,arr)

*return* (element+1)

})

*console.log*(p)

**Filter() ->** Filters an array and form a new array which passes a given test. Return an array.

letarr2=[1,3,9,4,6,98,100]

lett=*arr2.filter*((*item*)=>{

*return* item>10

})

*console.log*(t)

**reduce() ->** itreturns a value and perform an operation.

letarr3=[1,3,9,4,6,98,100]

letk=*arr3.reduce*((*n1*,*n2*,*n3*)=>{

*return* n1+n2\*n3

})

*console.log*(k)

1. Note: The **console.log** statement is placed after the **return** statement, which means it will never be executed. The **return** statement ends the function execution and returns a value to the caller. Any code after the **return** statement is unreachable.

**Javascript in Browser:**

Form JS was initially created to make webpages alive. Js was written right in the webpages`s HTML to make it look attractive. The browser has an embedded engine calked JavaScript engine or JS runtime. JS ability in the browser is very limited to protect the user’s safety.

HTML is the skeleton of web pages. JS is the logic of webpages. CSS is the styling in webpages.

Developer’s Tool: every browser have some developers tool which make a developer life little bit easy. There are three important tools that are elements, console and network.

Element: all HTML elements

Console: Where all the error are logged.

Network: All return request

**JS console Object**

The console object has several methods and log is one of them. Some of them are as follows:

Assert() –

Clear() – clear the console

Log()- output a message to console

Table()- displays table data

Warn()- use for warning

Error()- use for error

Info()- use for special information

Time()- shows time

**Alert, Prompt, Confirm in JS:**

alert() – used to invoke a mini window with a message. Eg- alert(“how are you”)

prompt() – it takes an input and store it, it take user input as a string always

prompt(variable, “No”) – No is a default value here, it can store default values also.

document.write(varibale)

confirm () – shows a message and waits for the user to press it either ok or cancel it. Return true for ok and false for cancel.

**Window object, BOM,DOM:**

Shows we have the following when Javascript runs in a browser. Window object represents browser window and provides methods to control them. It is also called global object in JS.

JS CORE

BOM

DOM

**BOM:** The **Browser Object Model** (BOM) is used to interact with the browser. It represents the additional objects provided by the browser for working with everything except documents.

The default object of browser is window means you can call all the functions of window by specifying window or directly. For example:

1. window.alert("hello javatpoint");

is same as:

alert("hello javatpoint");

You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth

**DOM:** Document object model represents the page content as HTML.

document.body – page body as JS object

document.body.style.backgroud = “green” – change the background colour to green