

JavaScript

Q. What is JavaScript?

- * JavaScript is a light weight / text-based programming language.
- * It is most well known as scripting language for Web pages.
- * What can we do with JS ?
 - * Web development
 - * Mobile application development
 - * Game development

Firefox uses **SpiderMonkey** JS engine.

Chrome uses **V8** JS engine.

Adding JS in Code

- You can add JS code in HTML document by using the `<script>` tag that wraps around JavaScript code.

The `<script>` tag can be used in the `<head>` or `<body>` section in the HTML.

Example

```
<html>
```

```
    <head>
```

```
        <head>
```

```
        <body>
```

```
            <script src="index.js"></script>
```

```
        </body>
```

```
</html>
```

You will have to create a javascript file with .js extension.

Variables

A variable is a container for storing data (storing data values).

4 ways to declare a Javascript Variable:

- * Using `var`
- * using `let`
- * Using `const`
- * we can also declare a variable without using any keyword.

Ex

```
var x = 5;
```

```
let y = 6;
```

```
z = 10;
```

When to use "const"?

When we declare any value with "const" keyword then it means those value are constant and cannot be changed.

```
const price = 5;
```

```
let a = 5;           number literal
```

```
let name = "Rishabh"; string literal
```

```
let status = true; boolean literal
```

Variable Naming Rules

- ⇒ Cannot be a reserved keyword.
- ⇒ Cannot start with a number.
- ⇒ Cannot contain space or !-!.
- ⇒ Should be meaningful.
- ⇒ Use camelcase. Ex- let arrName;

Primitive Types

In JavaScript there are 7 primitive data types:

- * string `let s = 'Rishabh';`
- * number `let n = 15;`
- * bigint It is a integer with arbitrary precision.
- * boolean `let y = true;`
- * undefined `let a;`
- * symbol a data type whose instance are unique and immutable.
- * null `let z = null;`

Dynamic Typing

JavaScript has dynamic types. This means that the same variable can be used to hold different data types.

Ex `let n; // undefined`

~~`n = 5; // a number`~~

~~`n = 'Rishabh'; // a string`~~

Reference Types

- * Objects
- * Arrays
- * Functions

Objects

A Javascript object is a collection of named values.

Object values are written as **key : value** pair.

Objects are variables too, but it contains multiple values.

Ex

```
let person = {  
    fName: 'Rishabh',  
    lName: 'Kushwaha',  
    age: 20  
};
```

Access in Objects

⇒ person.age

⇒ person['age']

Arrays

An array is a special variable, which can hold multiple values of multiple data types or of same type.

Ex

```
let name = ['Rishabh', 'Babbar', 'Lakshay'];
```

```
let person = ['Rishabh', 20, 'BCA'];
```

Operators

* Arithmetic

* Assignment

* Comparison

* Bitwise

* Logical

Arithmetic

+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division

% Modulus

++ Increment

-- Decrement

Assignment

=

+=

-=

%=

*=

**=

Comparison

>	greater than	$= = =$	equal value & type
<	less than	$! = =$	not equal value & type
\geq	greater than equal to	$! =$	not equal
\leq	less than equal to	$= =$	equal to
?			ternary operator

Logical

AND $\&\&$

OR $||$

NOT $!$

Bitwise

Bitwise AND $\&\&$

Bitwise OR $|||$

Control Statements

* if - else

* switch

if - else

```
if (condition) {  
    // code  
}  
else if (condition) {  
    // code  
}  
else {  
    // code  
}
```

single } single } multiple

Switch Case

```
switch (Expression) {  
    case @: [ ] ;  
        break ;  
    case : [ ] ;  
        break ;  
    default: [ ] ;  
}
```

Loops

It is used when we want repetition of task.

- * for loop
- * while loop
- * do-while loop
- * for-in loop
- * for-of loop

For loop

for (; ;)

{

initialization condition iteration

// statement

y

While loop

let n = 0; initialization

while () condition

{

 // statement

i++ / i--; iteration

y

Do-while loop

```
let i=0; ← initialization
do
{
    ==== II statement
    i++; ← iteration
} while (↳ condition);
```

Break and Continue

The **break** statement "jumps out" of a loop.

The **continue** statement "jumps over" or "skip" one iteration in the loop.

* Creating an Object

```
const rectangle = {
```

length: 1,

breadth: 2,

; (H, II) a function that is created inside an object

II is known as a method.

```
draw: function() {
```

```
    console.log('lets draw');
```

}

y;

* Object Creation by -

* Factory Function

* Constructor Function

* Factory Function

```
function createRectangle(len, bre) {
```

```
    return rectangle = {
```

length: len,

breadth: bre,

; (Optional) psil baneos

```
draw() {
```

```
console.log('lets draw');
```

}

y;

y

if factory function calling

```
const rectangleObj = createRectangle(5,4);  
console.log(createRectangle(5,4));
```

y (function) : work

→ Constructor Function

In this function we will use pascal notation.

Pascal Notation → NumberOfStudents

Constructor function → properties (methods) →

it initialize / define object and doesn't return

without returning *

function Rectangle() {

'this' keyword represents the current
object we are working with.

this.length = 1,

this.breadth = 1

this.draw = function () {

 board : this board

 console.log('Drawing');

y (work)

Object creation using constructor

"new" keyword returns an empty object.

```
let rectangleObj = new Rectangle();
```

Empty object or example :-

* Dynamic Nature of Objects

Addition / removal of any property in the object is possible.

If creation

```
rectangleObj.color = 'yellow';
```

If deletion

```
delete rectangleObj.color;
```

* Constructor Properties

⇒ Functions are objects

In JS functions are also known as objects.

⇒ We can use '.' operator in any function and we will see many properties of it.

Types in JS

→ Primitive or value types

→ here ~~copy~~ is created

→ Reference or object types

→ here ~~address~~ is passed

* Primitive

→ Number

→ String

→ Boolean

→ Null

→ Symbol

* Reference

→ Functions

→ Objects

→ Arrays

* Iteration through Objects

→ for-of

→ for-in

* For-of

It is only used in iterables i.e. arrays and maps, etc.

Syntax

```
for (variable of iterable) {  
    // statement
```

* For-in

→ It is only used in objects.

→ It is used to check whether any property is present or not in the object.

Syntax

```
for (key in object) {  
    // statement
```

* Object Cloning

It is used to create a copy of original object with same properties included in it.

⇒ Using assign

Ex 11 original

```
const employee = {  
    name: 'Rishabh',  
    age: 20,  
    height: 5.8  
}
```

If cloning

```
const copyEmp = Object.assign  
({ y, employee});
```

```
console.log(copyEmp);
```

⇒ **Using iteration**

```
const object = {a: 1, b: 2, c: 3};
```

// using for-in loop

```
for (const property in object) {
```

```
    console.log(`$ ${property}; ${object[property]}`);
```

y // `` = backtick character

⇒ **Using spread operator (...)**

```
let copyEmp = {...employee};
```

```
console.log(copyEmp);
```

* Garbage Collector

- It is used to deallocate the memory.
- It runs automatically in background.
- We have no control over it, mean we cannot control the start/end of it.

* Garbage Collector

- It is used to deallocate the memory.
- It runs automatically in background.
- We have no control over it, means we cannot control the start/end of it.

* Math Object

Math is a built-in object that has properties and methods for mathematical constants and functions.

* Math Properties

The syntax is:

Math.property

Ex - Math.E will returns Euler's number.

Math.PI will returns PI

Math.SQRT2 will returns square root of 2

* Math Methods

The syntax is:

Math.method(number)

Example: Math.floor(3.7) = 3

Math.round(x) Returns x rounded to its nearest integer

Math.ceil(x) Returns x rounded up to its nearest integer

Math.floor(x) Returns x rounded down to its nearest integer

Math.trunc(x) Returns integer part of x

Math.pow(x,y) Returns the value of x^y to the power of y.

Math.sqrt(x) returns the square root of x.

Math.abs(x) Returns the absolute (true) value of x.

Math.min() Returns the lowest value in a list of arguments

Math.max() Returns the highest value in a list of arguments.

* String

The string object is used to represent and manipulate a sequence of characters.

Creating Strings

- It can be created as primitive, from string literals or as objects, using `String()` constructor.

As primitive

Ex 1) primitive

```
const str1 = 'A string primitive';
```

```
const str2 = "A string primitive";
```

```
const str3 = `A string primitive`;
```

Note: here ` are backtick characters

As objects

```
const str4 = new String("A string object");
```

- We can convert primitive string into object by using `.` character.

- A string object can also be converted to its primitive counterpart with the `valueOf()` method.

* String Methods

- ⇒ **length** It returns the length of a string
- ⇒ **slice()** Extracts a part of string and returns the extracted part in a new string
- ⇒ **substring(start, end)** Same as slice but start & end values are less than 0
- ⇒ **substr(start, end)** Same as slice but, second parameter specifies the length of the extracted part
- ⇒ **replace()** It replaces a specified value with another value in a string.
- ⇒ **toUpperCase()** A string is converted into upper case.
- ⇒ **toLowerCase()** A string is converted into lower case.
- ⇒ **concat()** It joins two or more strings
- ⇒ **trim()** It removes whitespace from both ends of a string.
- ⇒ **trimStart()** and **trimEnd()**

* Template Literals

It uses back-ticks (`) rather than the quotes ("") to define a string.

Ex

```
let text = `Hello World`;
```

⇒ You can use both single and double quotes inside a string.

Ex

```
let text = `He's often called "Johnny";`
```

⇒ Template literals allows multiline strings:

Ex

```
let text =
```

```
`This is also known as  
the best  
news ever`
```

* Interpolation

Template literals provide an easy way to interpolate variables and expression into string.

This method is called string interpolation.

Syntax

With `String` (Old fashioned way)

* Variable Substitution

Template literals allows variables in strings.

Ex

```
let fname = "Rishabh";  
let lname = "Kushwaha";
```

```
let text = `Welcome ${fname}, ${lname}!`;
```

Output

```
Welcome Rishabh Kushwaha!
```

* Escape Sequence

Special characters can be encoded using escape sequence.

Code

Result

Description

\0 null character

' single quote

" double quote

\ backslash

\n newline

\t tab space

* Date Objects

Date objects are created with the new Date() constructor.

There are 9 ways to create a new date object.

new Date()

new Date(date string)

// const d = new Date("October 13, 2014 11:13:00");

new Date(year, month)

new Date(year, month, day)

new Date(year, month, day, hours)

new Date(year, month, day, hours, minutes)

new Date(year, month, day, hours, minutes, seconds)

new Date(year, month, day, hours, minutes, seconds, ms)

new Date(milliseconds)

* Getter & Setter

⇒ get

It is a function without arguments. The Javascript getter methods are used to access the properties of an object.

⇒ set

A function with one argument. The Javascript setter methods are used to change the values of an object.

* Array Object

An array object is used to store multiple values in a single variable.

Ex

```
const cars = ["Saab", "Volvo", "BMW"];
```

* Array Operations

* Adding new elements

* Finding elements

* Removing elements

* Splitting elements

* Combining elements

* Array Creation

let arr1 = [1, 2, 3, 4, 5];

* Insertion

- at start -- arr.push(0); Index no. of values to be removed
- at middle -- arr.splice(1, 0, 'a', 'b');
- at end -- arr.unshift(6); Create last

* Searching

We can use:

indexof() - Searches an array for an element and returns its position.

includes() - checks if an array contains the specified element.

NOTE - Predicate Functions

Predicate functions are functions that return a single TRUE or FALSE.

* Removing

- at start -- arr.shift(); Delete first element
- at middle -- arr.splice(index, no. of elements you want to delete)

→ end -- arr.pop()

* Emptying An Array

arr1 = [];

|| array will be empty because of garbage collector

let num1 = [1, 2, 3]; num1 --> address for num1

let num2 = num1; num2 --> has to be

num1 = [];

|| but it's not a good way delete an array

|| because it's an object so address will be
copied, not the copy is created.

So num1 --> [1, 2, 3] address -> deleted.
num2 --> different address

So when we assign num1 = []

num1 --> [1, 2, 3] address -> deleted
num2 --> different address

Here, num1 will point to an empty array.

⇒ The best way to delete an array is
using the below code.

`arr.length = 0;`

⇒ There is another way to delete an array using the `splice()`.

`num1.splice(0, num1.length);`

* Combining And Slicing An Array

⇒ Combine

`let arr1 = [1, 2, 3];`

`let arr2 = [3, 4, 5];`

`let combine = arr1.concat(arr2);`

⇒ Slice

`combine.slice(starting index, ending index);`

if ending index means it will take value

until `index ending - 1`

Combining an array using Spread operator (...)

`let combine = [...arr1, ...arr2];`

Creating a copy using spread operator (...)

`let another = [...combine];`

* Iterating / Traversing an Array

We will study ~~for loop~~, because it works on iterables only.

```
for (let value of arr1) {  
    console.log(value);
```

y point with variable for iteration

⇒ forEach()

The forEach() method calls a function for each element in an array.

```
arr1.forEach(function (number) {
```

```
    console.log(number);
```

(y); pushes values, return primitive value, avoid error

return value, return object, avoid error

Converting forEach() into arrow (⇒) function

```
(...) into arr1.forEach(number => console.log(number));
```

* Joining Arrays

```
(...) into const joined = arr1.join(' ');
```

(join with ' ') -> string type

* Splitting an Array

Let's let mes = "This is my name"; If we do

```
let parts = mes.split(' ');
```

* Sorting an Array

```
let num = [3, 7, 0, 9, 6];
```

num.sort()

We can also sort any array in reverse order after sorting in an unsorted array.

[num.reverse()]

reverse() is used to reverse an array, it overwrites the original array.

⇒ Above functions cannot sort an object, it can be done using predicate function.

* Filtering an Array

⇒ The filter() method creates a new array

filled with elements that pass a test provided by a function.

⇒ It doesn't change the original array.

Syntax

array.filter(function([currentValue, index, arr],
↓ ↓
thisValue))

→ If thisValue is given then
it will ignore the value.

If Here → refers to optional values

Ex

let num = [5, 7, -6, 3, 0];

let filtered = num.filter(function(value) {

return value ≥ 0);

});

Using \Rightarrow function

let filtered = num.filter(value \Rightarrow value ≥ 0);

* Mapping in Array

→ map() creates a new array from calling a function for each array element.

→ It calls a function once for each element in an array.

→ It does not change the original array.

Ex

For example if we want to map all the elements of an array to their squares.

```
let num = [7, 8, 9, 1];
```

```
let items = num.map(function(value) {
```

```
    return 'student-no' + value;
```

```
});
```

Using \Rightarrow function

```
let items = num.map(value  $\Rightarrow$  'student-no' + value);
```

* Mapping With Objects

```
let num = [7, 8, -7, -10];
```

```
let filtered = num.filter(value  $\Rightarrow$  value  $\geq 0$ );
```

```
let items = filtered.map(function(number) {
```

```
    return {value: number};
```

```
y);
```

Using \Rightarrow function

```
let items = filtered.map(number  $\Rightarrow$  {value: number});
```

* Chaining

In chaining we replace a variable with its original code.

Ex let items = num
• filter(value \Rightarrow value ≥ 0)
• map(number \Rightarrow {value; number});

In the above code we replaced filtered

variable with its original code.

(Explain what is happening in the following code)

step 0 initial program

{1, -2, 3, 4} \rightarrow max

(Explain what is happening in the following code)

if (selectedNumbers.length == 0) {
 console.log("No numbers")
} else {
 console.log(`Selected numbers: \${selectedNumbers}`);
}

(Explain what is happening in the following code)

if

selectedNumbers.length

(Explain what is happening in the following code)

program

After clicking on step 0 the output is

• Step 0: nothing

Ex

```
let items = num  
    .filter(value => value >= 0)  
    .map(number => {value: number});
```

In the above code we replaced ~~filtered~~ variable with its original code.

* Functions

A block of code that fulfills a specific task is called a function.

Creation Syntax

```
function functionName() {  
    // body of function  
    statement  
    statement  
    ...  
}
```

Why we need functions?

- Better readability
- code reuse
- To avoid bulky code
- To reduce bugs

* Function Declaration

```
function run(parameters) {  
    console.log('running');  
}
```

* Function calling / invoking

```
run();
```

* Hoisting Concept

a. If we call the function before the actual function is written, will it run properly?

⇒ Yes, because in JavaScript all the functions are sent at the top of the file and this process is known as **Hoisting** and its done automatically by js engine.

* Types of function assignment

- i) Name function assignment
- ii) Anonymous function assignment

Named function Assignment

```
let stand = function walk() {  
    console.log('walk');  
}
```

y

Anonymous function Assignment

```
let stand2 = function() {  
    console.log('walk');  
}
```

y

Calling the function

stand()

If we call the walk() after the assignment

it will not be called, we can only

call it using stand().

Can we call the function before its declaration?

No, because hoisting only works with

function initialization (not with function

assignment.

JS is Dynamic Language

when let $n = 1$; now value of $n = 1$
then $n = 1$ always remains same throughout

No error will come in the above code.

```
function sum(a,b){  
    return a+b; }  
sum(1,2) → 3
```

console.log(sum(1)) → If we pass only one value
the output will be NaN (Not a Number)
because by default value 2 will be not
defined.

Now see code passing two values
console.log(sum()) → NaN
because both values will not be defined
console.log(sum(1,2,3,4,5)) → 3 because 1 → a
and 2 → b and rest of the numbers will go
to arguments.

Arguments are special of objects.

* REST operator (...)

It is also known as spread operator, it is used to concatenate & copy the arrays.

If in a function we have multiple parameters then we can handle all the parameters with the help of the rest operator.

```
function sum(num, value, ...args) {  
    console.log(args)  
}
```

```
sum(1, 2, 3, 4, 5, 6, 7, 8)
```

- ⇒ By using the rest parameter we can only assign the values before args not after it.
- ⇒ We can store the varying parameters in the array form using the rest operator.

* Default Parameters

When a user does not pass any value to the function then the function takes the value by itself.

⇒ Below r=5 is default parameter

So, if you pass a value to 'r' in the function calling then it will take the passed value otherwise it will take the default value.

```
function interest(p, r=5, y=10) {
```

```
    return p * r * y / 100;
```

y

```
console.log(interest(1000));
```

⇒ If 'r' is a default parameter then all the parameters to the right side of it must be also default. (It's a rule)

* Getter

Setter

getter → to access the properties

setter → to change or mutate properties

both get and set methods will provide meaningful code

let person = { first: 'Rishabh',

lastName: 'Kushwaha',

fullName: 'Rishabh Kushwaha' };

getter } get fullName() {

return `\${person.firstName} \${person.lastName}`;

setter } set fullName(value) {

let parts = value.split(' ');

this.firstName = parts[0];

this.lastName = parts[1];

}

((and firstName) not allowed)

console.log(person.fullName);

→ Rishabh Kushwaha

person.fullName = 'Love Babbar';

console.log(person.fullName);

→ Love Babbar

* Try and Catch Block

- ⇒ The try statement defines a code block to run (to try).
- ⇒ The catch statement defines a code block to handle any error.
- ⇒ The finally statement defines a code block to run regardless of the result.
- ⇒ The throw statement defines a custom error.

```
try {  
    Block of code to try  
}  
catch (err) {  
    Block of code to handle errors  
}  
finally {  
    Block of code to run regardless of the try/  
    catch result  
}
```

* Scope

Scope determines whether accessibility (visibility) of variable.

JavaScript has 3 types of scopes:

- Block scope
- Function scope
- Global scope

Ex

{

var n = 2;

let y = 2;

y

ii x can be used here

ii y cannot be used here

Variable declared with var keyword cannot have block scope.

* Reducing an Array

→ The reduce() method executes a reducer function for an array.

⇒ The reduce() method returns a single value: the function's accumulated result.

⇒ The reduce() method does not change the original array.

```
let arr = [1,2,3,4,5];
```

```
let sum = 0;
```

```
for (let value of arr) {  
    sum = sum + value;
```

```
y
```

```
console.log(sum);
```

→ 15

Now by using reduce() method with ⇒ function

```
let sum1 = arr.reduce((accumulator, currentValue) =>  
    accumulator + currentValue, 0);
```

/*→ accumulator is like the sum variable that stores the value.

→ currentValue is like the loop

→ 0 is initialization of the value (if 0 is not set then the accumulator will start from the first value i.e. 1)

```
console.log(sum1);
```

→ 15