# 1. Javascript Introduction

## * ECMAscript

→ ECMAscript is a standard on which Javascript is based.

→ It was created to ensure that different documents on javascript are actually talking about the same language.

→ Javascript & ECMAscript can almost always be used interchangably.

→ Javascript is very liberal in what it allows.

## * Execute Javascript

→ Javascript can be executed right inside one's browser you can open the javascript console and start writing javascript there, By using REPL ( Read- Evaluate - Print - Loop).

→ Another way to ~~use~~ execute javascript is a runtime ~~like~~ environment like Node.js which can be installed and used to run javascript code.

→ Yet another way to executed javascript is by instesting is inside <script> tag of an HTML document.

# 2. Variables & Datatypes

## * Variable

→ JS is called dynamically typed language which means datatypes of variable can be changed.

→ A variable is a container that stores a value. (A name of a storage location)

e.g. let a = 7;

assignment operator

literal

reserved keyword

identifier

## * Rules of Identifiers

→ Letters, digits, underscores, dollar ($) signs are allowed

→ Must begin with ~~$~~ _ ~~or~~ a letter

→ Reserved words of Javascript not allowed

→ JS is case sensitive language

→ Way of writing identifiers are recommended as below:

camelCase
snake_case
PascalCase

* var vs let vs const

→ Before ES6 (ECMAScript 6) var were used.

→ var is globally scoped while let & const are blocked scoped

→ var can be updated & re-declared within its scope.

→ let can be updated but not re-declared

→ const can neither be updated nor be re-declared

→ var variables are initialized with undefined whereas let and const variables are not initialized.

→ const must be initialized during declaration unlike let and var

* Datatypes (Primitive)

NULL ⇒ Intentional absence of value, to be explicitly assigned
NUMBER ⇒ positive or negative integers, floating numbers
SYMBOL
STRING
BOOLEAN ⇒ true or false
BIGINT
UNDEFINED

→ If you want to know datatype of any variable then use keyword typeof.

e.g let a = 50;
typeof a; output ⇒ 'number'

→ OBJECT (key-value pair) is non-primitive datatype

e.g let a = null;
let b = 345;
let c = true; or let c = false;
let d = BigInt("567");
let e = "PARTH";
let f = Symbol("I am a symbol");
let g = undefined; OR let g;

```
const item = {
    item1 : true,
    item2 : 50
}
```

output for : console.log(item[item1]); ⇒ true

OR

item.item1

→ String

→ Text or sequence of characters.

→ value can be written in " " OR ' '.

→ If value has " " then write value in ' '.

→ string are stored as at indices.

eg let name = "TONY STARK";

output for: name[0] ⇒ 'T'
name.length ⇒ 10
"TONY"[0] ⇒ 'T'
"TONY".length ⇒ 4
"TONY" + 1 ⇒ 'TONY1'

→ Typecasting string to number: Number.parseInt("20");

→ NaN ( Not-A-Number )

→ The NaN global property is a value representing Not-A-Number, but the datatype of NaN is 'number'.

eg 0/0 , NaN-1 , NaN * NaN , NaN+1 , ...
output ⇒ NaN

# 3. Operators

## * Arithmetic Operators

addition operator ⇒ +
subtraction operator ⇒ -
multiplication operator ⇒ *
division operator ⇒ /
remainder operator (Modulo) ⇒ %
power operator (Exponentiation) ⇒ **

## → Operator Precendence

$$( ) \rightarrow ** \rightarrow *, /, \% \rightarrow +, -$$

right to left in expression

left to right in expression

## * Assignment Operators

| | | | | |
|---|---|---|---|---|
| = | ⇒ | x = y | ⇒ | x = y |
| += | ⇒ | x += y | ⇒ | x = x + y |
| -= | ⇒ | x -= y | ⇒ | x = x - y |
| *= | ⇒ | x *= y | ⇒ | x = x * y |
| /= | ⇒ | x /= y | ⇒ | x = x / y |
| %= | ⇒ | x %= y | ⇒ | x = x % y |
| **= | ⇒ | x **= y | ⇒ | x = x ** y |

## * Increment & Decrement (Arithmetic Operator)

- ++ ⇒ a++ ⇒ a = a+1
- -- ⇒ a-- ⇒ a = a-1

a++ ⇒ post-increment ⇒ use then change
++a ⇒ pre-increment ⇒ change then use

## * Comparison Operators

== ⇒ equal to
=== ⇒ equal value and equal type
!= ⇒ not equal
!== ⇒ not equal value or not equal type
> ⇒ greater than
< ⇒ less than
>= ⇒ greater than or equal to
<= ⇒ less than or equal to
? ⇒ ternary operator

## * Logical Operators

&& ⇒ logical and
|| ⇒ logical or
! ⇒ logical not

## * Type Operators

typeof ⇒ Returns the type of a variable
instanceof ⇒ Returns true if an object is an instance of an object type

## * Bitwise Operators

& ⇒ AND ⇒ Sets each bit to 1 if both bits are 1
| ⇒ OR ⇒ Sets each bit to 1 if one of two bits is 1
^ ⇒ XOR ⇒ Sets each bit to 1 if only one of two bits is 1
~ ⇒ NOT ⇒ Inverts all the bits
<< ⇒ Zero fill left shift ⇒ Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>> ⇒ Signed right shift ⇒ Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>> ⇒ Zero fill right shift ⇒ Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

# 4. Concepts

## * Linking JS file

→ Make a file in same folder as HTML file, give it name "app.js".

→ To link javascript file with HTML write this code just above "</body>" tag.

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

## * Console.log()

→ To write (log) a message on the console

```
console.log("Apna College");   ⇒ Apna College
console.log(1234);             ⇒ 1234
console.log(2+2);              ⇒ 4
console.log("A", "B", "C");    ⇒ A B C
```

## * Template literals

→ Template Literals are used to add embedded expressions in a string.

```
let a = 5;
let b = 10;
console.log(`The total is ${a+b} Rupees.`);
console.log("The total is", (a+b), "Rupees");
```

back tick (acute) — back tick (acute)

} Both will give same output

→ Insert variables directly in template literal. This is called String Interpolation.

* Prompt

→ To take input from user on browser in a dialog box.

```
prompt("Enter your age...");
```

* Alert

→ To give (display) an alert message on browser in a dialog box.

```
alert("~~Enter~~ something is wrong");
```

* Truthy & Falsy values

→ In Boolean Context (meaning writing expression as a condition which will give result in boolean value), everything in JS is associated with true or false; but that doesn't mean value itself is true or false, they are treated as true or false in boolean context.

falsy values: false, 0, -0, null, undefined, "" (emty string), NaN, 0n (BigInt)

truthy values: Everything else

# 5. Conditional Statements

* if statement

```
if (expression) {
    // do something
}
```

expression result is true then will execute

* if-else statement

```
if (expression) {
    // do something
} else {
    // do something
}
```

* if... else if... statement / if... else if... else if... else statement

```
if (expression1) {
    // do something
} else if (expression2) {
    // do something
}
```

* Turnary Operator ( shorthand method for if-else statement)

```
expression ? // do something : // do something
```

if true  if false

* Switch statement

```
switch (expression) {
        answer
  case result1 :  // do something
                  break;

  case result2 :  // do something
                  break;

  default :       // do something

}
```

# 6. String

## Escape sequence characters (\)

→ If we want to write quote inside quote then we have write \ (backslash) before quote.

```
console.log("PARTH\"PORTFOLIO\"");  ⇒ PARTH"PORTFOLIO"
```

→ \n ⇒ new line  \t ⇒ tab  \r ⇒ carraige return

## trim() Method

→ Remove whitespace from starting and ending of a string.

```
let msg = "      Hello      ";
let newMsg = msg.trim();
console.log(newMsg.length);   ⇒ 5
console.log(msg);             ⇒ Hello
console.log(newMsg);          ⇒ Hello
console.log(newMsg.toUpperCase());  ⇒ HELLO.
console.log(newMsg.toLowerCase());  ⇒ hello.
```

function (method)

property

## Immutable String

→ No changes can be made to strings.

→ Whenever we do try to make a change new string is created and old one remains same.

```
newMsg[0] = 'p';   ← This is not possible
```

## * indexOf() Method

→ Returns the first index of occurence of some value in string or gives -1 if not found.

```
let str = "ILoveCoding";

str.indexOf("Love");   ⇒ 1
str.indexOf('J');      ⇒ -1  (not found)
str.indexOf('o');      ⇒ 2
```

## * Slice() Method :

→ Returns a part of the original string as a new string

starting index

```
str.slice(5);   ⇒ Coding
str.slice(1,5);   ⇒ Love
```

not included end index

str.slice(-num); or str.slice(length - num);

-1

first 11+1 ⇒ 8

* replace() Method

→ Searches the value in the string & returns a new string with the value replaced.

str.replace("Love","do"); ⇒ I do Coding
str.replace('o', 'x'); ⇒ I Lxve Coding

* repeat() Method

→ Returns a string with the number of copies of a string

str.repeat(3); ⇒ I Love Coding I Love Coding I Love Coding

* concat() Method

→ combine strings

str.concat("Yes", "No", str); ⇒ I Love Coding Yes No I Love Coding

* includes() Method

→ Returns boolean value if given string is in main string

str.includes("I Love Coding"); ⇒ true

* startsWith() Method

→ str.startsWith("I"); ⇒ true

* endsWith() Method

→ str.endsWith("ng"); ⇒ true

* toLocaleString() Method

→ 10000.toLocaleString("en-IN"); ⇒ 10,000

# 87. Array

→ Collection of things

→ let fruits = ["banana", "apple", "mango"];
let num = [2, 4, 6, 9];
let arr = ["parth", false, 7];

arr.length ⇒ 3
arr[0] ⇒ 'parth'
fruits[1].length ⇒ 5
fruits[0][2] ⇒ 'n'

* Mutable Array

→ num[2] = 7;
console.log(num); ⇒ (4) [2, 4, 7, 9]

→ num[10] = 5;
console.log(num); ⇒ (11) [2, 4, 7, 9, empty × 6, 5]

* Datatype of Array is Object.

* toString() Method

→ let arr = [11, 12, "parth", 10];
console.log(arr); ⇒ (4) [11, 12, 'parth', 10]
console.log(arr.toString()); ⇒ 11,12,parth,10

## * join() Method

→ Joins all array elements using a separator

```
let num = [2, 4, 6, 8];
let n = num.join("_");
console.log(n);   ⇒   2_4_6_8
typeof n;   ⇒ string
```

## * push() Method

→ Adds a new element at the end of an original array (and returns updated length)

```
let num = [2, 4, 6, 8];
num.push(10);  ⇒ if wrap it with "console.log" ⇒ 5
num.push("parth");  ⇒ console.log(num.push("parth"));  ⇒ 6
console.log(num);  ⇒ (6)[2, 4, 6, 8, 10, 'parth']
```

## * pop() Method

→ Removes the last element of an original array and return it (element).

```
let num = [2, 4, 6, 8, 10, "parth"];
let c = num.pop();
console.log(c);   ⇒ 'parth'
console.log(num);  ⇒ (5)[2, 4, 6, 8, 10]
```

* unshift() Method

→ Adds a new element at the start of an original array (and returns updated length)

```
let num = [2, 4, 6, 8];
num.unshift(10);
console.log(num); ⇒ (5) [10, 2, 4, 6, 8]
```

* shift() Method

→ Removes the start element of an original array and returns that element.

```
let num = [2, 4, 6, 8];
console.log(num.shift()); ⇒ 2
console.log(num); ⇒ (3) [4, 6, 8]
```

* delete Operator

```
let num = [2, 4, 6, 8];
delete num[3];
console.log(num); ⇒ (4) [2, 4, 6, empty]
```

* reverse() Method

→ Reverse an original array.

```
let num = [5, 6, 7, 8, "abc", 3, "bcd"];
num.reverse();
console.log(num); ⇒(7)['bcd', 3, 'abc', 8, 7, 6, 5]
```

* sort() Method

→ Sort an original array alphabetically

```
let num = ["bac", "7", 7, 9, 8, 9, 4, "abc", 1, 3, "ghy"];
num.sort();
console.log(num); ⇒(11)[1, 3, 4, '7', 7, 8, 9, 9, 'abc', 'bac', 'gh
```

* 
```
let num = ["bac", "7", 77, 99, 8, 9, "abc"];
num.sort();
console.log(num); ⇒(7)['7', 77, 8, 9, 99, 'abc', 'bac']
```

→ If we gives a compare function as an argument to sort() then we can sort as per ascending or descending ~~order~~ order.

```
let compare = ( a , b ) ⇒{ return a - b; }
let num = [ 5 , 12 , 124, 22, 55 , 1 ];
num.sort(compare);
console.log(num); ⇒ (6) [ 1 , 5 , 12 , 22 , 55 , 124 ]
```

* splice() Method

→ Removes / replaces / adds elements in an original array and returns deleted items

```
splice( start , deleteCount , item0 ... itemN )
```

```
let num = [ 2 , 4 , 6 , 8 , 10 , 12 ];
~~num splice~~
console.log (num.splice(4)); ⇒ (2) [ 10 , 12 ]
console.log(num); ⇒ (4) [ 2 , 4 , 6 , 8 ]

console.log (num.splice(0,1)); ⇒ [2]
console.log(num); ⇒ (3) [ 4 , 6 , 8 ]

console.log(num.splice( 0 , 3 ,"black","white")); ⇒[3][4,6,8]
console.log(num); ⇒ (2) [ 'black' , 'white' ]

console.log (num.splice(1,0,"grey")); ⇒ []
console.log(num); ⇒ (3) [ 'black' , 'grey' , 'white' ]

console.log(num.splice(1,1, "red")); ⇒ ['grey']
console.log(num); ⇒ (3) [ 'black' , 'red' , 'white' ]
```

## * slice() Method

→ Copies a portion of an array

```
let num = [ 2 , 4 , 6 , 8 ];
console.log(num.slice(2));   ⇒ (2) [6, 8]
console.log(num.slice(2,3)); ⇒ [6]
console.log(num.slice(-2));  ⇒ (2) [6, 8]
console.log(num);            ⇒ (4) [2, 4, 6, 8]
```

## * indexOf() Method

→
```
let arr = ["red , "yellow", "blue"];
console.log(arr.indexOf("yellow"));  ⇒ 1
console.log(arr.indexOf("green"));   ⇒ -1 (not found)
console.log(arr.indexOf("Yellow"));  ⇒ -1 . (not found)
```

## * includes() Method

→
```
let arr = ["red", "yellow", "blue"];
console.log(arr.includes("red"));    ⇒ true
console.log(arr.includes("Blue"));   ⇒ false (not found)
console.log(arr.includes("green"));  ⇒ false (not found)
```

* concat() Method

→ Merge 2 or more arrays. Returns new merged array.

```
let arr1 = [ "red", "blue", "green" ];
let arr2 = [ 2, 4, 6, 8 ];
let arr3 = arr1.concat(arr2);
console.log(arr1);  ⇒ (3) [ 'red', 'blue', 'green' ]
console.log(arr2);  ⇒ (4) [ 2, 4, 6, 8 ]
console.log(arr3);  ⇒ (7) [ 'red', 'blue', 'green', 2, 4, 6, 8 ]
```

* Array References

→ Reference ⇒ address in memory

→ Array variables are reference variable which stores addresses of values.

[1] == [1] ⇒ false ⇒ because both array are stored in different addresses

```
let num1 = [1, 2, 3];
let num2 = [1, 2, 3];
num1 == num2          ⇒ false

num2 = num1
num1 == num2          ⇒ true
num2.push(4);
console.log(num1);    ⇒ (4) [ 1, 2, 3, 4 ]
```

## * Constant Array

→ values can be changed but not the address of values

```
const arr = [1, 2, 3, 4];
arr.push(5);
console.log(arr);  ⇒ (5) [1, 2, 3, 4, 5]

arr = [1, 2, 3];  ⇒ This is not possible
```

## * Nested Array

```
let arr = [ [2, 4] , [5, 6] , [7, 8] ];
console.log(arr[1]);  ⇒ (2) [5, 6]
```

# 8. Loops

→ Used to iterate (repeat) a piece of code.

for - loops a block of code no. of times
for-in - loops through the keys of an object
for-of - loops through the values of an object
while - loops a block of code based on a specific condition
do-while - while loop variant which runs atleast once

## * For Loop

→
```
for( initialisation;   condition;   updation) {
        // do something
}
```

eg.
```
for (let i=0; i<=5; i++) {
        console.log(i);
}
```

Output:
```
0
1
2
3
4
5
```

→ initialisation is executed one time; after checking the condition (based on condition - true or false) loop body is executed. then updation happens and again condition is checked ...

e.g. odd numbers print from 1 to 10:

```
for (let i=1; i<=10; i+=2){
    console.log(i);
}
```

output: 1
3
5
7
9

e.g. even numbers print from 1 to 10:

```
for( let i=2; i<=10; i+=2) {
    console.log(i);
}
```

Output: 2
4
6
8
10

e.g. sum of first n natural numbers:

```
let sum=0;
let n= Number.parseInt(prompt("Enter the value of n"));
for (let i=1; i<=n; i++){
    sum += i;
}
console.log(`Sum of first ${n} natural numbers
            is ${sum}`);
```

output: n=10 ⇒ Sum of first 10 natural numbers is 55.

e.g. multiplication table of number n:

```
let n=Number.parseInt(prompt("Enter a number");
for( let i=1; i<=10; i++) {
    console.log(`${n} x ${i} = ${i*n}`);
}
```

Output: n=5 ⇒ 5 × 1 = 5
5 × 2 = 10
⋮

## * Nested loop

eg
```
let n = 5;
let str = "";
for ( let i = 1; i <= n; i++ ) {
    for (let j = 1; j <= i ; j++) {
        str += j ;
    }
    str += "\n" ;
}
console.log( str );
```

Output :
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

## * while loop

```
while ( condition ) {
    // do something
}
```

⇒ If condition never becomes false, the loop will never end

eg
```
let i = 1 ;
while ( i <= 5 ) {
    console.log(i);
    i++ ;
}
```

Output :
```
1
2
3
4
5
```

→ Returns Greatest Integer :

```
num = 12345;
num = Math.floor(num/10);
```
⇒ num = 1234

* break keyword.

→ Used to get out of the loop execution

e.g.
```
let i=1;
while (i <= 5) {
    if (i == 3) {
        break;
    }
    console.log (i);
    i++;
}
```

output: 1
2

* do-while loop

e.g.
```
let p1 = "abc123";
let p2 = "";
do {
    p2 = prompt("Enter your password");
} while( p1 != p2 );
```

Output: first at least once. prompt asks for p2 then checks the condition; if false then loop ends and if true then again prompt will appear.

## * Loops with Arrays

→ let fruits = ["mango", "apple", "banana", "orange", "litchi"];

```
for( let i = 0;  i < fruits.length;  i++) {
    console.log(`${i+1}. ${fruits[i]}`);
}
```

output:
1. mango
2. apple
3. banana
4. orange
5. litchi

## * For-of Loop

iterable (arrays & strings)

→ for( element of collection ) {
    // do something
}

write anything like a, b, c

eg. for( char of "PARTH") {
    console.log(char);
}

output: P
A
R
T
H

eg for( fruit of fruits ) {
    console.log(fruit);
}

output: mango
apple
banana
orange
litchi

* Simple To-do App

→ HTML page

"list" - to show all todos
"add" - to add a todo
"delete" - to delete a task
"quit" - to exit the todo app

→ JS

```
let todo = [];
let req = prompt("please enter your request");

while (true) {

    if (req == "quit") {
        console.log("Quitting App");
        break;
    }

    if ( req == "list") {
        for ( let i=0; i< todo.length; i++) {
            console.log(`${i+1}. ${todo[i]}`);
        }
    } else if ( req == "add") {
        let task = prompt("enter task");
        todo.push(task);
        console.log("task added");
```

```
} else if ( req == "delete" ) {
    let index = Number.parseInt( prompt("Enter task index"));
    todo.splice (index-1 , 1);
    console.log("task deleted");
} else {
    console.log ("wrong request");
}

req = prompt("please enter your request");
}
```

# 9. Object Literals

→ Used to store keyed collections & complex entities

→ Objects are a collection of properties
property ⇒ ( key, value ) pair

→ When printing an object order of properties may not be same as stored.

eg
```
let delhi = { latitude : " 28.7041° N",
              longitude : " 77.1025° E"
};
```

eg
```
const student = { name : "parth",
                  age : 27 ,
                  marks : 95 ,
                  city : "surat
};
```

Output : { name : 'parth', age : 27, city : 'surat', marks : '95' }

eg
```
const post = { username : "@parth",
               content : "This is my post",
               likes : 150 ,
               reposts : 5 ,
               tags : [ "@abc", "@cdef" ]
};
```

* Get Value

e.g. const student = { name: "parth",
marks: 80
};

```
console.log ( student["name"] );
console.log ( student.name );
```

key
key

output: 'parth'
'parth'

e.g. ~~const~~ let prop = "name";
console.log( student[prop]);

output: 'parth'

* JS automatically converts objects keys to strings. Even if we made the number as a key, the number will be stored as string

e.g. const obj:={ 1 : 'a' ,
2: 'b',
true: 'c',
null : 'd',
undefined: 'e'
};

console.log(obj[1]); ⇒ 'a'
console.log(obj[null]); ⇒ 'd'
Console.log(obj.2); ⇒ error: not possible
console.log(obj.null); ⇒ 'd'

## * Add / Update Value

```
e.g const obj = { name: "parth",
                  age: 50,
                  marks: 90,
                  city: "Surat"
};
~~student~~
obj.city = "Mumbai";      // Update
obj.gender = "Male";      // add
obj.marks = "A";          // update
~~obj~~
delete obj.age;           // delete
console.log(obj);
```

Output : { name : 'parth', marks: 'A', city: 'Mumbai', gender: 'Male' }

## * Nested Objects

```
e.g const classInfo = { parth: { grade: "A+",
                                 city: "Surat" },
                        om: { grade: "A",
                              city: "Delhi" },
                        hardik: { grade: "B",
                                  city: "Mumbai" }
};
console.log (classInfo.parth); => { grade: 'A+', city: 'Surat' }
console.log (classInfo.om.city); => 'Delhi'
classInfo.om.city = "pune";
console.log (classInfo.om.city); => 'Pune'
```

## * Array of Objects

e.g.
```
const classInfo = [ { name: "parth",
                      grade: "A+",
                      city : "Surat" },
                    { name "om",
                      grade "A",
                      city "Delhi" },
                    { name "hardik",
                      grade "B",
                      city "Mumbai" }
];
console.log(classInfo); ⇒ [ {...}, {...}, {...} ]
console.log(classInfo[0]); ⇒ { name: 'parth', grade... }
console.log(classInfo[1].city); ⇒ 'Delhi'
classInfo[1].city = 'Pune';
console.log(classInfo[1].city); ⇒ 'Pune'
```

## * Math Object

→ console.log(Math); ⇒ Properties and Methods

→ Properties ⇒ Math.E ⇒ 2.71...
Math.PI ⇒ 3.14...
Math.SQRT2 ⇒ 1.4142...
Math.SQRT1_2 ⇒ 0.7070...
...

Methods ⇒ Math.abs(n) ⇒ returns positive
Math.pow(a,b) ⇒ $a^b$
Math.floor(n) ⇒ nearest smallest int. (round off to <=n)
Math.ceil(n) ⇒ nearest largest int
Math.random() ⇒ value from 0 to 1 ; 0 included but 1 is not.
⋮

## * Random Integers

e.g. From 1 to 10 random numbers

```
let num = Math.random();          ⇒ 0.999...
num = num * 10;                   ⇒ 9.99...
num = Math.floor(num);            ⇒ 9
num = num + 1;                    ⇒ 10
Math.floor((Math.random() * 10) + 1);
```

end start

e.g. For range min = 21 & max = 25

start

```
Math.floor((Math.random()*(max-min+1))+min);
```

If you want last number included