→ History of Java Script

The early to mid-1990s was an important time for the Internet.

Key players like Netscape and Microsoft were in the midst of browser wars, with Netscape's Navigator and Microsoft's Internet Explorer going to head to head.

In September 1995, a Netscape programmer named Brandon Eich developed a new scripting language in just 10 days. It was originally named Mocha, but quickly became known as ~~Java~~ Live Script and later, Java Script.

→ Why Java Script

It's a most popular programming Language.
It has been ranked the most commonly used programming language.
Java Script is the programming Language of the Web.
Java Script is easy to learn.

→ What is Java Script ?

Java Script is a scripting language that is one of the three core languages used to develop websites.

* Java Script lets you add functionality and behaviours to your website, allowing your websites visitors to interact with content in many imaginative ways
* Java Script is primarily a client-side language, meaning it runs on your computer within your browser. However, more recently the introduction of Node.js has allowed Java Script to also execute code on servers.

→ How to open Java Script in VS code

Desktop → open new folder as JS

→ In Google → click on left button → In that click on Inspect → Click on Console

Eg: Console.log(10+20);

O/P : 30 [O/P will come directly]

→ In VS code → Path select → New file → Save on desktop → filename as file.js → Save

Open file → write the programme → save it → Run the programme

if the programme is not executed → click on left button → In that open Integrated terminal

NOTE:

→ error will occur in the path mistake (or) spelling mistake of the document.

→ Without saving the document we run means it will not give O/P it will show blank screen.

Eg: Console.log(10);

terminal → node file.js → O/P : 10

→ Programming Language : Computer programme is a sequence of instruction within using a computer programming language to perform a specific task by the computer.

Act of writing this program is called Computer Programming

1. Java
2. Python
3. C++
4. Ruby
5. Perl
6. PHP
7. Java Script

→ Keywords

Keywords are those reserved words which will have a pre-defined meaning defined in the programming language.

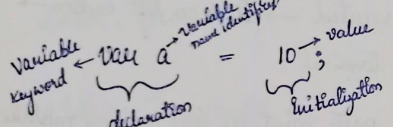
Keywords should be written only in lower case.

var x → variable declaration

x = 10 → initialization

var x = 10 → declaration + initialization

* x is the undefined variable



* while reinitialization it is not restricted the same value we can change the value by different values.

If we type var x = "hello"; → it is re-declaration.

→ Values and Variables

We store the value of the program inside one variable.

Variables are used to store the data.

Syntax: var variable_name = "value";

→ Rules to declare Identifier (address)

1. The first character must be a letter [or] an underscore (_) or an dollar ($).
2. You can't use a number as the first character.
3. The rest of the variable name can include any letter, any number, or the underscore or dollar. Can't use any other special characters, including space.
4. Variable names are case sensitive.
5. No limit to the length of the variable name.
6. You can't use one of JavaScript's reserved words as a variable name.

Ex: var name, var_name, var $namE, n420, n_name, n $name.

→ Data types

2 types of data types

1. Primitive datatype
   1. Number
   2. String
   3. Boolean
   4. Undefined
   * Bigint
   * Symbol
2. Non primitive data type (reference)
   1. object

1. Number

```
var x = 10;
console.log(x);
```

o/p : x = 10

```
var x = 10;
console.log(typeof x);
```

o/p : number

* typeof is a operator to check the variable.
* we can store single character in single or double quotes it will not show error in execution.

→ Keywords

Keywords are those reserved words which will have a pre-defined meaning defined in the programming language.

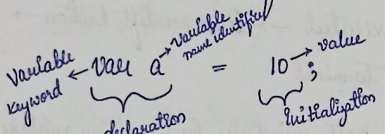
Keywords should be written only in lower case.

var x → variable declaration

x = 10 → initialization

var x = 10 → declaration + initialization

⊛ x is the undefined variable



⊛ while reinitialization it is not restricted the same value we can change the value by different values.

If we type var x = "hello"; → it is re-declaration.

→ Values and Variables

We store the value of the program inside one variable.

Variables are used to store the data.

Syntax: var variable_name = "value";

→ Rules to declare Identifiers (address)

1. The first character must be a letter [or] an underscore (_) or an dollar ($).
2. You can't use a number as the first character.
3. The rest of the variable name can include any letter, any number, or the underscore or dollar. Can't use any other special characters, including space.
4. Variable names are case sensitive.
5. No limit to the length of the variable name.
6. You can't use one of JavaScript's reserved words as a variable name.

Ex: var name, var_name, var $name, n420, n_name, n$name.

→ Data types

2 types of data types

1. Primitive datatype
   1. Number
   2. String
   3. Boolean
   4. Undefined
   * BigInt
   * Symbol
2. Non primitive data type (reference)
   1. object

1. Number

```
var x = 10;
console.log(x);
```

o/p : x = 10

```
var x = 10;
console.log(typeof x);
```

o/p : number

* typeof is a operator to check the variable.
* we can store single character in single or double quotes it will not show error in execution.

2. String : Inside the double quote if it is single or double quote of group of char, number, special char.
* It is not differentiate b/w single or double quote it is string.

Eg: var x = " " ;
console.log (x);
o/p empty string

3. Boolean : In boolean we don't use any single [or] double quotes.

Eg: var x = true ;
console.log (x);
o/p true

4. Undefined : default value for one variable is undefined

Eg: var x ;
console.log (typeof x) ;
o/p undefined

→ Operators : It is used to do the operation on operands
* process is called operation.
* special char we use to do operation is operands
* Values we are using is called as operands

1. Arithmetic operator : It is used to do arithmetic operation

+, - , * , / , % , exponential (**) , increment (++), decrement (--)

Combination of datatype and operator

1. Num + Num = Num
2. Num + String = String
eg: var a = 4 var a+b
var b = hello o/p 4 hello string
3. Num + boolean = Num
4. Num + undefined = (Nan)

2. Assignment operator :
It is used to assign a value for a variable. storing it in the left side value.

assigning (=)
addition assigning (+=) a = a+b or a += b
subtraction assigning (-=) a = a-b or a -= b
multiplication assigning (*=) a = a*b or a *= b
division assigning (/=) a = a/b or a /= b
modulus assigning (%=) a = a%b or a %= b
exponential assigning (**=) a = a**b or a **= b

Eg: var a = 10 ;
var b = 20 ;
a = a+b ; a += b ;
console.log (a) ;

3. Comparision operator : It is used to do compare
* comparision result is always a boolean & all the time

equals to (==) compare only value not the datatypes
triple equals (===) compare value as well as datatypes
not equal (!=) compare only value not the datatypes
not double equal (!==) compare value as well as datatypes
less than equal (<=)
more than equal (>=)
lesser than (<)
greater than (>)

Eg: var a = 10;
var b = 20;
console.log(a==b);

O/p value are equal → true
value are not equal → false

var a = 10
var b = "100"
console.log(a==b); // Check only values
console.log(a===b); // Check the values as well as datatypes

var a = 10;
var b = 20;
console.log(a!=b);
console.log(a!==b);

4. Logical operator : Used to combine multiple boolean to produce one boolean value.

AND (&&) → when we want satisfy all condition
OR (||) → when we want to satisfy any one condition

→ Flow Control statements

Control statements are used to control the execution flow some line of codes in the program.

Here we have 2 types of control statements

1. Decision making statements
2. Looping statements

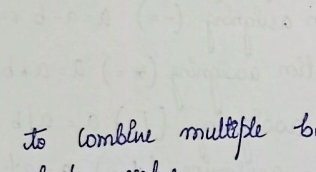
1. Decision making statements :

Decision making statements are the statements used to execute the set of lines based on some conditions.

1) a) if b) if else c) if else if else d) Switch case

a) if : if body will be executes only when the condition is true.

Flow chart of if condition



Syntax of if statement

```
if (condition)
{
  Statements;
}
```

Eg: if (true)
{
console.log ("this is if body");
}

O/p: Condition is true if executed.
Condition is false if it is not executed.

① print the no if only if it is greater than 10.

var a = 13;
if (a>=10)
{
console.log(a);
}

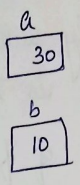
O/p = 13

→ The process of swapping the values by using third variable

```
var a = 10;
var b = 20;
var temp = b;
b = a;
a = temp;
```

a: 20
b: 10
temp: 20

→ The process of swapping the value without third variable




```
var a = 10;
var b = 20;
a = a+b;
b = a-b;
a = a-b;
```

② print the number if it is even number

```
var num = 12;
if (num % 2 == 0)
{
  console.log (num);
}
```

③ print the number if it is odd number

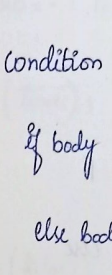
```
var num = 13;
if (num % 12 == 1) (OR) (num % 2 != 0)
{
  console.log (num);
}
```

b) if else : If the condition is true, if body will be executed. If the condition is false, else body will be executed.

Syntax :

```
if (condition)
  {
   console.log ("this is if body");
  }
 else
  {
  console.log (" this is else body");
  }
```

Flow chart



① Write a program to check the given number is even no or odd no

```
var num = 57;
{
 console.log {"
                if (num % 2 == 0)
                {
                 console.log (" the given no is even");
                }
                else
                {
                 console.log (" the given no is odd");
                }
```

```
var num = 58;
② if (num % 2 == 0)
 {
  console.log (num + " is even no ");
 }
 else
 {
  console.log (num + " is odd no ");
 }
```

③ print true if the given number is divisible by 23. If not print else.

Algo I/p var num

O/p T F

num is divisible by 3 else

if (num %. 23 == 0)

```
var num = 126;
if (num %. 23 == 0)
{
 console.log (true);
}
else
{
 console.log (false);
}
```

④ print true if the given non negative number is 1 or 2 more than a multiple of 20.

Algo I/p var num

O/p T F

~~num~~ { divide the num by 20 else

if rem is 1 or 2

if (num %. 20 == 1 || num %. 20 == 2)

```
var num
if (num %. 20 == 1 || num %. 20 == 2)
{
console.log (true);
}
else
{
console.log (false);
}
```

⑤ print true if the non negative number is 1 or 2 less than multiple of 20. So example 38 and 39 return true, but 40 return false

Algo I/p var num

O/p T F

divide by 20

if rem 18 or 19 else

if (num %. 20 == 19 || num %. 20 == 18)

```
var num = ;
if (num %. 20 == 19 || num %. 20 == 18)
{
 console.log (true);
}
else
{
 console.log (false);
}
```

⑥ Love 6

No 6 is a truly great number. Given 2 int values a, b. print true if either one is 6. or if either sum or diff is 6.

Algo I/p two a, b

O/p T F

a is 6 or b is 6 or if either a+b is 6 or diff of a & b is 6 — else

```
var a, b
if (a == 6 || b == 6)
{
  console.log (true);
}
else if (a+b == 6 || a-b == 6)
{
  console.log (true);
}
else
 {
  console.log (false);
 }
```

⑦ Given three ints a, b, c return true if one of them is 10 or more less than one of the others

⑧ print true if given value is in the range 10 to 20?

```
var a = 113;
console.log (a >= 10 && a <= 20 ? "a is in range" :
                "a is not in range");

// condition ? statement 1 : Statement 2;
// when condition is true statement 1 will be executed
// when condition is false statement 2 will be executed
```

```
if (a >= 10 && a <= 20)
{
  console.log (true);
}
else
{
  consol.log (false);
}
```

→ Switch case : It is used only in equal and comparision statements.

```
eg: var day = "Sunday";
    switch (day)
    {
      case "Monday" : console.log ("Weeday");
      case "Tuesday" : console.log ("Weekday");
          —||—
          —||—
      case "Saturday" : console.log ("Weekend");
    }
eg: var day = "Wednesday";
    switch (day)
    {
      case "Monday" : console.log ("Weekday");
        break;
      case "Tuesday" : console.log ("Weekday");
        break;
          —||—
          —||—
    }
```

→ Looping statement

Looping statements can be used to executed some set of statements repeatedly for multiple time.

1. for loop
2. while loop
3. dowhile loop
4. for in loop
5. for off loop

1. For loop : when we know the start to end condition clearly we can use for loop.

Work : Statements start of for will be executed values unless the end condition is true and once end condition becomes false for loop will be terminated.

Syntax :
```
for (startcondition ; end condition ; counter)
{
  statement ; // execute until end condition is false
}
```

→ Programmes

1. print the numbers 1 to 10

```
for (var a = 1 ; a <= 10 ; a++)
                           (or) a = a + 5
{
  console.log (a);
}
```

2. print the numbers 25 to 1

```
for (var a = 25 ; a >= 1 ; a--)
                           (or) a = a - 5
{
  console.log (a);
}
```

3. Print the table value of 3 from 1 to 10

```
for (var a=1 ; a<=10 ; a++)
{
  console.log (a * 3) ;
}
```

4. Print the table value of given variable from 1 to 10

```
var x = 5
for (var a=1 ; a<=10 ; a++)
{
  console.log (a * x) ;
}
```

5. Print the table value of given variable from 1 to 10 use concatination operation

```
var x = 7 ;
for (var a=1 ; a<=10 ; a++)
{
  console.log ("7 * " + a + " = " + a * x) ;
                [OR]
              (x + " * " + a + " = " + x * a) ;
}
```

2. While loop :

* While loop can be used when we doesn't know exact number of iterations.
* It will execute the statement repeatedly unless the condition becomes false.

Syntax :

```
while ( num >= n)
{
  num = num / n
  n++
}
```

→ programmes

1. Print all the digits in the given number in "Reverse"

— given number 7869

```
var num = 7869
while (num > 0)
{
  var last = num % 10 ; // take last digit
  console.log ( last ) ; // add last to previous sum
  num = num / 10 ; [OR] num = parseInt (n / 10) ; // deleting last digit
}
```

ParseInt is used to ignore the decimal number (OR) values.

eg: console.log (ParseInt (5.7596)) ;

O/P = 5

→ Another method to delete last number

```
var num = 7869
while (num > 0)
{
  var last = num % 10 ;
  console.log (last) ;
  num = (num - last) / 10 ;
}
```

last → num % 10 → print last → num = num / 10

2. To find the factorial of 40320

```
var num = 40320 ;
var n = 1 ;
while (num >= n)
{
  num = num/n ;
  n++ ;
}
console.log (num);
```

$\frac{40320}{1} \rightarrow \frac{40320}{2} \rightarrow \frac{20160}{3} \rightarrow \frac{6720}{4}$

$\frac{1680}{5} \rightarrow \frac{336}{6} \rightarrow \frac{56}{7} \rightarrow \frac{8}{8}$

3. Count the number of digits in the given number

```
— var num = 1234 ;
  var count = 0 ;
  while (num > 0)
  {
    num = parseInt (num/10);
    count++ ;
  }
  console.log (count);
```

O/P = 4

4. print the sum of all the digits in the given number

```
— var num = 1234 ;
  var sum = 0 ;
  while (num > 0)
  {
    var last = num % 10 ;
    console.log (num);
    sum = sum + last ;
    num = parseInt (num/10);
  }
  console.log (sum);
```

take last → add last to sum → delete last

5. Reverse the given number 1234

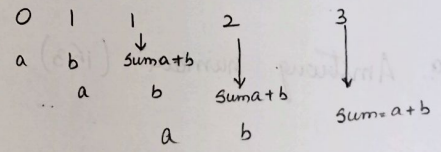
```
— var num = 1234 ;
  var rev = 0 ;
  while (num > 0)
  {
    var last = (num % 10) ;
    rev = (rev * 10) + last ;
    num = parseInt (num/10);
  }
  console.log (rev);
```

6. Check the given number is palindrome or not.

```
— var num = 101 ;
  var temp = num ;
  var rev = 0 ;
  while (num > 0)
  {
    var last = num % 10 ;
    rev = (rev * 10) + last ;
    num = parseInt (num/10);
  }
  num = temp ;
  console.log (rev == num ? "palindrome" : "not a palindrome");
```

7. print the fibanocci series upto given number.

```
— var num = 13 ;
  var a = 0 ;
  var b = 1 ;
  var sum = 0 ;
  console.log (a);
  console.log (b);
```



```
while (sum < num)
{
  sum = a+b;
  console.log (sum);
  a = b;
  b = sum;
}
```

8. Print the frequency of the given digit in the number.

```
var num = 11231;
var digit = 1;
var count = 0;
while (num > 0)
{
  var last = num % 10; // taking out last digit
  if (last == digit) // compare last to given digit
  {
    count++;        // if same make a count
  }
  num = parseInt (num/10); // delete last digit
}
console.log ("The digit" + digit + "is repeated" + count +
             "times");
```

9. Amstrong number (153)

3. Do while : will execute the statement even though the while condition atleast once.

Behaviour

* execution is done first and then condition will be decided.
* We can use do while whenever we want to execute of statements.
* Even though condition is false at the initial stage.

~~Functions~~

// check the given num is present in the num1 or not.

```
var n = 13465;
var n2 = 8;
var count = 0;
var count1 = 0;
while (n2 > 0)
{
  var last = n2%10;
  count1++;
  n2 = parseInt (n2/10);
  while (n > 0)
  {
    var last1 = n%10;
    if (last == last1).
    {
      count++;
      break;
    }
  }
```

Syntax

```
do
{
  Body;
} while (condition/end)
```

```
if (count == count1)
{
  console.log ("number is present");
}
else
{
  console.log ("number is not present");
}
```

→ Functions

Function in javascript is a block of code.
Function is used to ~~reuse~~ reuse the code again & again for multiple inputs.

Syntax:
```
function function_name (parameters)
{
  statements;
}
function_name (arguments)
```
function declaration

function invoking or calling

Functions can be declared one single time & can be called [or] invoked any number of time.

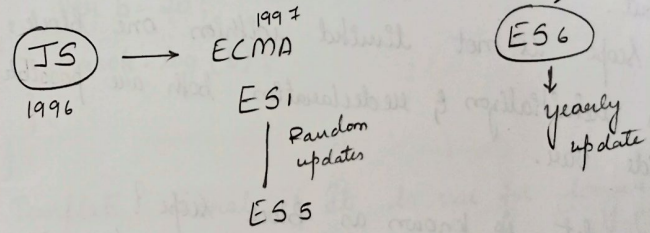
This function can be parameterized or o parameter function.

Eg: function to add two numbers

```
function add (num1, num2)
{
  sum = num1 + num2;
  console.log (sum);
}
add (1,2);
add (29,27);
```

Section 2

→ In 1996 javascript was created.
→ In 1997 handed over to ECMA and resulted to ECMA script.
→ In 2015 ES6 decided to use
→ After 2015 year changing the version upto 2020 it was ES11.



→ ES6
① keywords → let, const
② template literal
③ default parameters
④ Arrow function
⑤ Destructuring
⑥ Rest operators
⑦ Spread operator

Section 2 Concepts
① Array
② String
③ Function & types
④ Return keyword.

→ Difference b/w var, let and const

	Declaration & Initialization	Reinitialize	Redeclaration	Scoping
Var	we can declare & initial in diff line	we can re-initial by var keyword	we can redeclare by var keyword	function
let	—11—	—11—	we can't redeclare by let keyword	block
const	we can't declare & initial in diff line	we can't reinitial by const keyword	—11—	block

1. Why var is known as function scope?
* Because outside of our block if we will try to access the value of this block we are getting this output.
* The scope is not limited within one block.
* Even reinitialization & redeclaration both are possible inside var.

2. Why let is known as Block scope?
* Because outside of a block if we will try to access the property of this block we will get error.
* Because the scope is limited within one block.
* We cannot access the property of outside of the block.
* Only re-initialization is possible in let keyword re-declaration is not possible.

3. Why const is known as Block scope?
* Because outside of a block if we will try to access the property of this block we will get error.
* Because the scope is limited within one block.
* We can't access the property of outside of the block.
* Only initialization & declaration is possible in const keyword.

```
function example ()
{
  var a = 10;
  if (a <= 10)
  {
    var b = 20;
    console.log (b);
  }
}
```

→ Template literal : It is use for longer concatination.

Syntax : `` `String ${variable} String` ``

→ Default arguments :
It is concept introduced in ES6, so the we can assign a default value for the parameters declared in the function.

If we donot pass an arguments while invoking a function then default value will be accessed.

```
eg: function add (a=1, b=1, c=1)
    {
        const sum = a+b+c;
        console.log (sum);
    }
    add (10, 20);
```

→ Array : Array is variable which can store multiple values in one variable. To declare array we have array litral called [].

Syntax : var arrayName = [value1, value2, ...]

```
eg: var x = [10, "hello", true, undefined, NaN, null]
    console.log (x);
```

Array in javascript can hold multiple values.
Array in javascript is heterogenous.
Array in javascript is not fixed in size.
Array in javascript is flexible.
Array size can be accessed by length property (arrayName.length).

Array values will be stored in one block & each and every values are stored separately inside the buckets & those buckets are identified by index positions. arrayName [index].

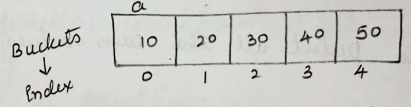
Index starts from 0 and end at length-1.

→ How to access an array

To access entire array we can call array name.
To access array values seperately, we can use index positions.

eg: arrayName [index].

```
var a = [10, 20, 30, 40, 50]
```



index = length - 1

```
// var a = [10, 20, 30, 40, 50];
   a[5] = 60;
   console.log (a);

// var a = [10, 20, 30, 40, 50];
   console.log (a.length);
```

① How to itterate an array

Array can be itterated by using a for loop starting from 0 and ending at array.length-1

```
for (var i=0; i<=a.length-1; i++)
{
    console.log (a[i]);
}
```

② WAFT print all the values of an array.

```
function printAll (arr)
{
  for (let i=0; i<= arr.length-1; i++)
  {
    console.log (arr[i]);
  }
}
printAll ([10, 20, 30, 40, 50]);
```

③ Write a function to print all the even values of an array.

```
function printEven (a)
{
  for (let i=0; i<a.length; i++)
  {
    if (a[i] % 2 = 0)
    {
      console.log (a[i]);
    }
  }
}
printEven ([1,2,3,4,5,6,7,8,9,10]);
```

④ WAFT print the sum of all the values in an array.

```
function sumOfAll (a)
{
  let sum = 0;
  for (let i=0; i<a.length; i++)
  {
    sum = sum + a[i];
  }
  console.log (sum);
}
sumOfAll ([1,2,3]);
sumOfAll ([1,2,3,4,5,6,7,8,9]);
```

⑤ WAFT print the count number of even numbers & odd numbers in a given array.

```
function countEvenOdd (a)
{
  let countEven = 0;
  let countOdd = 0;
  for (let i=0; i<a.length; i++)
  {
    a[i] % 2 = 0 ? countEven++ : countOdd++;
  }
  console.log (`The given array has ${countEven}
                even numbers`);
  console.log (`The given array has ${countOdd}
                odd numbers`);
}
countEvenOdd ([1,9,12,24,123,31,64,234]);
```

⑥ WAFT find a given value in an array.

```
function find (a, value)
{
  let result = -1;
  for (let i=0; i<a.length; i++)
```

```
    {
        if (a[i] == value)
        {
            result = i;
            break;
        }
    }
    console.log(result);
}
find([2,1,4,3,8,7,5,8], 8);
```

⑦ WAFT find the max value in a given array.

```
function findMax(a)
{
    let max = 0;
    for (let i=0; i<a.length; i++)
    {
        if (a[i] > max)
        {
            max = a[i];
        }
    }
    console.log(max);
}
findMax([2,5,8,4,3]);
```

⑧ WAFT sort an array in an ascending order.

```
function sort(a);
{
    for (let i=0; i<a.length-2; i++)
    {
        for (let j=0; j<a.length; j++)
        {
            if (a[i] > a[j]
            {
                var temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    return a;
}
console.log([3,1,4,5,2]);
```

⑨ WAFT .sort an array in descending order

```
function sort(a)
{
    for (let i=0; i<=a.length-2; i++)
    {
        for (let j=0; j<a.length; j++)
        {
            if (a[i]< a[j])
            {
                var temp = a[i];
                a[i] = a[j];
                a[j] = temps;
            }
```

```
    }
  }
  return a;
}
console.log([3,1,4,5,2]);
```

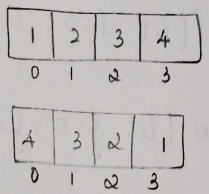
⑩ WAFT find the n^{th} largest number in array.

```
function nth largest (a, n)
{
  var x = sort (a);
  console.log (x);
  var max = x [x.length - n]
}
}
return a;
}
nth largest ([13, 8, 16, 21, 9], 2);
```

~~⇒ Array Inbuilt methods~~

⑪ WAFT reverse an array.

a = [1, 2, 3, 4]




```
function reverse (a)
{
```

→ Array Inbuilt methods

1. pop ( ) - deletes the last number [or] element. (Change original array)
2. push (argument) - add elements at last return (no of argument, length) (Change argument array)
3. shift ( ) - deletes the first element.
4. unshift (arr) - add elements at first. (no of arg)
5. includes ( ). - Return boolean value saying value is present or not.
6. IndexOf (argument) - Return index value if present & if not return -1
7. Reverse ( ) -
8. Sort ( ) -
9. Slice (start, end); -
10. Splice (start, count) (no of arg)

```
var a = [10, 20, 30, 40, 50]
a.splice (1, 3);
console.log (a);
```
[10, 50]

```
a.splice (0,1);
console.log (a);
```
[20, 30, 40, 50]

var a = [10, 20, 30, 40, 50]

a.splice(2, 0, 25);

console.log(a);

[10, 20, 25, 26, 30, 40, 50];

a.splice(2, 1, 25);

console.log(a);

11. Fill (value, start, end) - fill the index position

a.fill(10, 1, 93) → [1, 10, 10, 4, 5]

12. toString(- console.log(a.toString())

→ Functions and it types

Normal function | Anonymous function | Arrow function

function fun_name(p) | function

{

}

fun_name(arg)

→ Array inbuilt methods using Arrow function

1. forEach var a = [1, 2, 3, 4, 5, 6] (a.length - 1)

a.forEach((a) ⟹ { console.log(a) });

a.forEach((val, ind, z) ⟹ { console.log(a) });

var a = [10, 20, 30, 40, 50, 60]

a.forEach((val, index, array) ⟹ { console.log(a) }); sum = sum + val; console.log(sum);

2. filter — will accept the call back

a.filter(val, index, array)

3. Map

4. Some

5. Every

6. Reduce

* a.length is a property we not to use braces
* a.pop() will remove the last number

→ IndexOf

var a = [1, 2, 3, 4, 5, 6];

console.log(a.lastIndexOf(3));

console.log(a)

→ ~~Reverse~~ Slice

var a = [1, 2, 3, 4, 5, 6]

console.log(a.slice(3)); (3, 5)

console.log(a)

* it will do the returning a value
* from 3rd index it will print upto end - 1.

[4, 5, 6]

→ Loops

function slice(a, start, end)

→ for of → var a = {1, 2, 3}

for (const x of a)

{

console.log(x);

}

→ O/P 1 2 3

→ for in → var a = {a: 10, b: 20, c: 30}

for (const key in a)

{

console.log(key);

}

splice - we can do add, delete & replace

```
var a = [10, 20, 30, 40, 50, 60];
a.splice(a);
console.log(a);
```

** Assignment
Write logic for splice

filter → a.filter((val) => { return val % 2 == 0 });
as the name

Map →

some → Any return value true give result as True

way → a.map(e =>

→ Strings
we can write in double quote or single quote

```
var str = "String value";
```

① WAFT print every characters of the string.

```
function printchar(str)
{
  for (let i=0; i<str.length; i++)
  {
    console.log(str[i]);
  }
```

```
printchar("my name is Amrutha G.C");
```

② WAFT count the words

```
function printchar(str)
{
  let word = 0;
  for (let i=0; i<str.length; i++)
  {
    if (str[i] == " ")
    {
      word++;
    }
  }
  console.log(str != " " ? word+1 : word);
}
printchar("my name is Amrutha");
```

③ WAFT print the words in the string.

```
function printid(str)
{
  let word = "";   str = str + " ";
  for(let i=0; i<str.length; i++)
  {
    if (str[i] != " ")
    {
      word = word + str[i]
    }
    else
    {
      console.log(word);
      word = "";
    }
  }
}
printchar("my name is Amrutha");
```

When char is not space concat to word

④ WAFT find the given word in the string

```
function find (str , word)
{
  let w = " ";
  str = str + " " ;   var res = false ;
  for (let i=0 ; i<= str.length ; i++)
  {
    if (str [i]!= " ")
    {
      w = w + str[i] ;
    }
    else
    {
      if (w == word)
      {
        res = true ;
      }
      w = " "
    }
  } console.log (res);
}
find (" white board marker pen ", "board ");
```

⑤ WAFT

```
function find (str , word)
{
  var count = 1 ;
  var res = (" word is not present ");
  str = str + " " ;
  let w = " " ;
  for (let i=0 ; i< str.length ;i++)
{
  if (str [i] != " ")
  {
    w = w + str [i];
  }
  else          // char is a space so word has formed
  {
    if (w == word) // formed word is it same as given word
    {
      res = " word is present in ${count} position ;
      break ;
    }
    w = " "   // each time reassigning w to empty
    count ++;  // count the no of words
  }
}
  console.log (res);
}
find (" white board marker pen ", " marker ");
```

⑥ WAFT find the given sub string in the string.

```
function find (str , sub str)
{
  var res = (" word is present ");
  str = str + " "
  let w = " " ;
  for (let i=0 ; i<=str.length ; i++)
{
  let new Str = " "
  for ( let j=i ; j < (i + sub Str.length) ; j++)
  {
```

"my name is Amma"

" ame " → 3 ← 0 " my_ "
→ 1 " y_n "
→ 2 " _na "
→ 3 " nam "
→ 4 " ame "

res=true
break;

```
        newStr = newStr + str[j];
    }
    console.log(newStr);
  }
}
find("my name is Amrutha", "my na");
```

Another method

```
function find(str, subStr)
{
  let res = false;
  for(let i=0; i<str.length; i++)
  {
    let newStr = ""
    for(let j=i; j<(i+subStr.length); j++)
    {
      newStr = newStr + str[j];
    }
    if(newStr == subStr)
    {
      result = true;
      break;
    }
  }
  console.log(result);
}
find("my name is Ammu", "name");
```

m y n
0 1 2
. y n —

→ Another method

```
function find(str, subStr)
{
  let result = "not present";
  for(let i=0; i<str.length; i++)
  {
    let newStr = ""
    for(let j=i; j<(i+subStr.length); j++)
    {
      newStr = newStr + str[j];
    }
    if(newStr == subStr)
    {
      result = `The given substring ${subStr} is
                present from ${i} index position`;
      break;
    }
  }
  console.log(result);
}
find("my name is Ammu", "mmu");
```

Assignment

① String = my name is Ammu → ["my", "name", "is", "Ammu"]

② String = my name is Ammu → ["my-", "ame_is_am", "mu"]

→ WAFP find the longest word in the given string.

var max word = " " " my name is Amrutha"
var max lngth = 1 ↓ ↓ ↓ ↓
2 4 2 7

```
function largest (str)
{
 var max word = " ";        var word = " ";
 var max lngth = 0;         str = str + " ";
 for ( let i=0 ; i < str.length ; i++ ){
 if ( str[i] != " ")
  {
    word = word + str[i];
  }
 else
  {
   if (word length > max length)
    {
      max length = word.length;
      max word = word;
    }
    word = " ".
  }
 }
  console.log (max word);
}
largest (" my name is Amrutha");
```

→ WAFP find the smallest word in the given string.

```
function smallest (str)
{
 var min word = " ";
 var min length = str.length;
 var word = " "
 str = str + " ";
for ( let i=0; i < str.length ; i++)
 {
  if (str[i] != " ")
  {
   word = word + str[i];
  }
  else
   {
    if (word length < min length)
    {
      min length = word length;
      min word = word;
    }
    word = " "
   }
 }
  console.log (min word);
}
smallest (" my name is gunda")
```

→ WAFT to

```
function createArray (str)
{
  str = str + " "
  var word = " " ;
  var strArray = [ ];
  var j = 0;
  for (let i = 0; i < str.length ; i++)
  {
    if (str [i] != " ")
    {
      word = word + str [i];
    }
    else
    {
      strArray [j] = word ;
      j++ ;
      word = " " .
    }
  }
  console.log (strArray);
}
createArray ("Longer cap off-time");
```

→

```
function createArray (str, splitter)
{
  str = str + splitter;
  var word = " " ;
  var strArray = [ ];
  var j = 0;
  for (let i = 0; i < str.length ; i++)
  {
    if (str [i] != splitter).
    {
      word = word + str [i];
    }
    else
    {
      strArray [j] = word ;
      j++ ;
      word = " " .
    }
  }
  console.log (strArray);
}
createArray ("Longer cap off-time").
```

→ Strings in-built methods

var str = "hello everyone";

1. CharAt — console.log (str.charAt(1));

2. CharCodeAt - console.log (str.CharCodeAt(5));

3. Concat - console.log (str.Concat ("how are you"));

4. EndsWith - console.log (str.endsWith ("one"));
5. StartsWith - console.log (str.startsWith ("hell"));
} both will give o/p in boolean

6. Includes - console.log (str.includes ("hello")); → it will check whether it is present or not

7. IndexOf - console.log (str.IndexOf ("ello")); → It will not tell only whether it is present or not & it will tell the index value

→ Assignment

Logic for StartsWith & EndsWith

8. LastIndexOf - console.log (str.lastIndexOf ("name"));

9. Repeat - console.log (str.repeat (20)); → repeat the sentence 20 times.

10. Replace - console.log (str.replace ("my", "My")); → change the word in the sentence

11. Substring - console.log (str.substring (3,7)); → will accept 2 strings start to end-1

```
function find (str, substr)
{
  let result = false;
  for (let i=0; i< str.length ; i++)
  {
    let newStr = str.substring(i, i+substr.length);
    if (newStr == substr)
    {
      result = true;
      break
    }
  }
  console.log(result);
}
find
```

12. toLowerCase - console.log (str.toLowerCase());

13. toUpperCase - console.log (str.toUpperCase());

14. trim - console.log (str.trim()); → trim the original string length deleting the start & end void space.

15. trimEnd - console.log (str.trimEnd()); → delete the end void space.
console.log (str.trimEnd().length);

16. trimStart - console.log (str.trimStart()); → delete the starting void space
length

→ Destructuring:

```
var a = [10, 20, 30, 40, 50];
var [x, y] = a;
console.log (x); → x = 10
console.log (y); → y = 20
```

To replace word in string

```
function replace (str, word1, word2)
{
  var s = " ";
  var index = 0;
```

→ Rest operator:

```
var [x, y, ... z] = a;
```

→ Spread operator:

```
var a = [10,20,30];   var b = [40, 50, 60];
var c = [... a, ... b];
```

→ Higher order function: The function which accepts another as an argument is called higher order function.

→ First order function: The function which is sent as an argument for another function is called first order function.

a.filter ((v, i, a) ⇒ { })

filter → HOF; function as an argument → FOF

Section 3

6/7/22

OOPS

→ Object: Any entity which has its own properties and behaviours is called as object.

→ Object in Javascript: It is a collection of key value pairs.

var laptop = { Brand: "hp", Price: "10000" }

Brand → key; "hp" → value

→ Why do we require object in JS?

— To introduce an entity to the virtual world we can use objects in JS.

→ How to create objects in JS?

We can create the objects in JS by using 3 diff ways

1. Object literal
2. Constructor function
3. Class

// var laptop = ["HP", 10000, "1TB", "12GB"];

var laptop = { brand: "HP", price: 10000, storage: "1TB" }

we can keep as variable (or) keep as string

console.log(laptop);

console.table(laptop); → give o/p as in table form

1. Object literal: By using object literal we can declare an object by creating multiple key value pairs by separating them using coma (,).

How to access an object & its key (properties)

2. key: key in an object can be a normal variable or even a string.

3. Values can be anything.

object Name

var laptop = { "brand-name": "HP", → String
price: 10000, → Number
storage: "1TB" →
RAM: "12GB" →
"front-camera": true → boolean
"back-camera": undefined → undefined
sensors: ["proxy", "fingerprint"] → array
features: { faceLock: "yes" } → object
createDocs: () => {^} → function
xyz: Nan → Nan
abcd: Null → Null

Datatype

console.log("creating");

→ Higher order function: The function which accepts another as an argument is called higher order function.

→ First order function: The function which is sent as an argument for another function is called first order function.

a.filter ((v,i,a) ⇒ { })

filter → HOF; function as an argument → FOF

Section 3

6/7/22

OOPS

→ Object: Any entity which has its own properties and behaviours is called as object.

→ Object in Javascript: It is a collection of key value pairs.

```
var laptop = { Brand : "hp" , Price : "10000" }
               key     value
```

→ Why do we require object in JS?

— To introduce an entity to the virtual world we can use objects in JS.

→ How to create objects in JS?

We can create the objects in JS by using 3 diff ways

1. Object literal
2. Constructor function
3. Class

```
// var laptop = ["HP", 10000, "1TB", "8GB"];
var laptop = { brand : "HP", price : 10000, storage : "1TB" }
```

we can keep as variable (or) keep as string

```
console.log (laptop);
console.table (laptop);
```

give o/p as in table form

1. Object literal: By using object literal we can declare an object by creating multiple key value pairs by separating them using coma (,).

How to access an object & its key (properties)

2. key: key in an object can be a normal variable or even a string.
3. Values can be anything.

```
                    objectName
var laptop = { "brand-name" : "HP",      → String
               price : 10000,            → Number
               storage : "1TB"           →
               RAM : "8GB"               →
               "front-camera" : true     → boolean
               "back-camera" : undefined → undefined
               sensors : ["proxy", "fingerprint"] → array
               features : { faclock : "yes" } → objects
console.log("creating");
createDocs = () => { } → function
               xyz : Nan → Nan
               abcd : Null → Null
```

→ How to access an object and its key (properties)

When we key in normal variable we can (.) dot operator

1. Entire object can be accessed by using object Name
2. When object key is a normal variable we can use dot operator and call the property.
   Syntax : object . property.
3. When object key is a string mannered we can use array literal and call the property.
   Syntax : object [ ]
   - or [ ] { Number, String, boolean, undefined, Nan, null
     objects, array
   - key ( ) function
4. When key is a property access it by using . or array literal but without braces.
5. When key has a values of function, call it by using . operator & invoke that method by using ( ).

update : Even after declaration we can add a new property
→ We can add a new object properties

```
var obj = { x:10, y:20 } ;
delete obj.x;
obj.z = 30;
console.log (obj);
o/p : { y:20; z:30 };
```

Syntax : object.newproperty = value;

→ How to update an object properties.

1. Access the object properties and reassign.
   object.property = new value ;

→ Deleting an object properties
By using delete operation and accessing that object property we can delete the same object property.
Syntax : delete object.property

→ Math objects methods

```
var x = { a:10, "b-b":20, c:()=>{ console.log(30)} };
console.log (x.a);   (x.b[ ]),
```

① console.log (Math.abs (-20)); o/p : 20
abs - absolute → whether it is +ve -ve give o/p as +ve value [or] return absolute value.

② sqrt : return square root value for every numbers.
console.log (Math.sqrt (16)); o/p: 4

③ Cbrt : return cube root for every numbers.

④ ~~exp~~ pow() : return power of some number.

(**) console.log (Math.pow(a, 2));

⑤ sign(): return -1 if negative or +1 if positive numbers

console.log (Math.sign(a));

⑥ 圆周率 PI : return PI value.

console.log (Math.PI);

⑦ Random() : value b/w 0-1

console.log (Math.random());

⑧ Max() : return the max value out of passed numbers.

console.log (Math.max(a, b, 19, 20, 100));

⑨ Min() : return the min value out of passed numbers

⑩ Round() → round to next int if decimal >= 0.5 else round to previous int

console.log (Math.round(19.4));

Logic

```
function round (num)
{
  const dec = num%1;
  if (dec < 0.5)
  {
    num = num - dec;
  }
  else
  {
    num = num - dec+1;
  }
  console.log (num);
}
round (15.3);
```

⑪ floor() → round to previous Integer number.

console.log (Math.floor(24.9));

⑫ Ceil() → round of to next Integer number.

console.log (Math.ceil(24.9));

⑬ trunc() → removing the decimal number. return the integer part number.

console.log (Math.trunc(24.8986));

⑭ parseInt →

→ Date object :

Syntax : let d1 = new Date();

console.log (d1);

① Zero arguments ② Arguments

→ Methods

① Get ② Set ③ To

It is used to — used to change the value & set the values.

```
console.log (d2.getFullYear());
console.log (d2.getMonth());
console.log (d2.getDay());
console.log (d2.getDate());
console.log (d2.getHour());
console.log (d2.getMin());
console.log (d2.getSec());
console.log (d2.getMilliSec());
console.log (d2.getTimeZoneOffset());
```

8 + 8 + 1 (Time zone)
IST UTC

→ set → used to set the values.

```
d2.setFullYear (2024);
d2.setMonth (12);
d2.setDate (24);
```

7 + 7
IST UTC

→ To → convert the values to string manner & give the o/p as in string manner only.

```
console.log (d1.toString());          toLocalString
console.log (d1.toDateString());      toLocalDateString
console.log (d1.toTimeString());
```

3 (Std format) + 3 (Local format)

→ object creation by using 3 different process

① By using { }
② Constructor function
③ By using the class

① Constructor function :

```
function constructorFunction (brand, color, price)
{
  this.brand = brand;
  this.price = price;
  this.color = color;
}
var bike1 = new constructorFunction ("bajaj", "black", 200000);
console.log (bike1);
var bike2 = new constructorFunction ("RE", "black", 300000);
```

this keyword it will point current object address

→ By using the class * Functions inside the class & object are called as methods

→ we don't need to declare the function keyword inside the class body.

→ Constructor function is a special type of method because it is executed automatically whenever we create an object of the class. Since it is executed automatically, it takes a constructor & create the properties for our object

→ Class is a blue print of an object using this we can create any no of models

Inside the class we can declare multiple properties & methods

```
Class Bike
{
    Constructor ( brand, color, price)
    { this.newKey = value;
      this.brand = brand;
      this.color = color;
      this.price = price;
    }
    sample ( )
    {
      console.log (" this is sample function");
    }
}
var bike1 = ("bapy", "black", 100000)
var Bike1 = new Bike ( );
bike1.sample1( );
```

→ using the class we can create an object by using new keyword followed by class name.

Syntax : var obj = new ClassName ( );

Syntax

```
class Classname
{
  constructor ( parameters)
  {
    this.newproperty = parameter;
  }
  method1 ( )
  {
    stmts ;
  }
  method2 (parameters)
  {
    stmts ;
  }
```

Syntax

```
Class Classname
{
  constructor (parameters)
  {
    this.new property = parameter 1 ;
    this.new property = parameter 2 ;
  }
  method1 ( )
  {
    Statements ;
  }
  method2 (parameters)
  {
    Statements ;
  }
}
```

→ Encapsulation :

→ Inheritance : ① Single

```
Class Watsapp1
{
  constructor (user)
  {
    this.user = user;
  }
  Send Message ( )
  {
    console.log (" send a message");
  }
  upload dp ( )
  {
    console.log (" profile pic update ");
  }
}
var gunda = new Watsapp ( );
```

```
class Watsapp2 extends Watsapp1
{
  constructor (user)
  {
    this.user = user;
  }
  send Message()
  {
    console.log("send a message");
  }
  send voice Message()
  {
    console.log("recording audio to send
                 voice nok");
  }
}
var w2 = new Watsapp2();
w2.send Message(); // through sub class we can access super class method
w2.send voice message(); // ——//—— sub class method
```

instead of writing again we will inherited by the class1 → inheritance

using extends keyword

② Multilevel

```
class watsapp3 extends watsapp2
{
  Voice call()
  {
    console.log("start a voice call");
  }
}
let w3 = new watsapp3("Nivangan");
console.log(w3.user);
w3.send message();
w3.send voice nusage();
w3.voice call();
```

③ Hierarchical :

→ super keyword

We have to write super keyword in constructor compulsorily.

super keyword will create an object of parent class

Section 4

→ How javascript engine works

1. Hoisting

What is hoisting?
- How variable hoisted?
- How funct^{n} hoisted?

↓

Create an memories / programme before execution process.

It is a default behaviour of JS by declaring var & func keyword to the top of the scope.

hoisting

↓ a

variable → undefined → value defaultly name as undefined. and ~~end~~ more

function b() → code copied → by hoisted by code coping

→ Execution context : It is a place where all javascript programmes create memory (saved) and execut^{n} process.

Memory phase

hoisting

a → undefined

b() → code copied

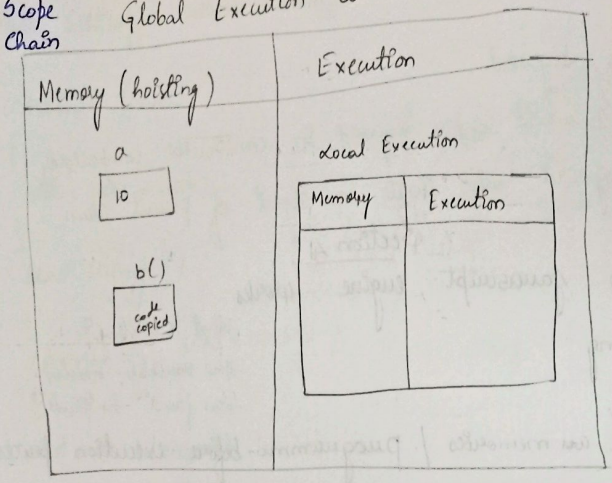
Execution process

a → undefined

b() → execute

→ Scope Chain

Global Execution Context

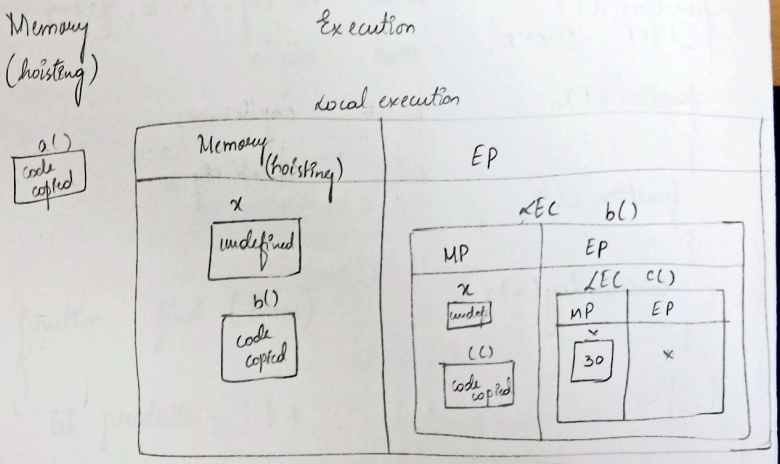



```
var a = 10;

function b()
{
  var x = 20;
  console.log(x);
}

console.log(a);

b();
```

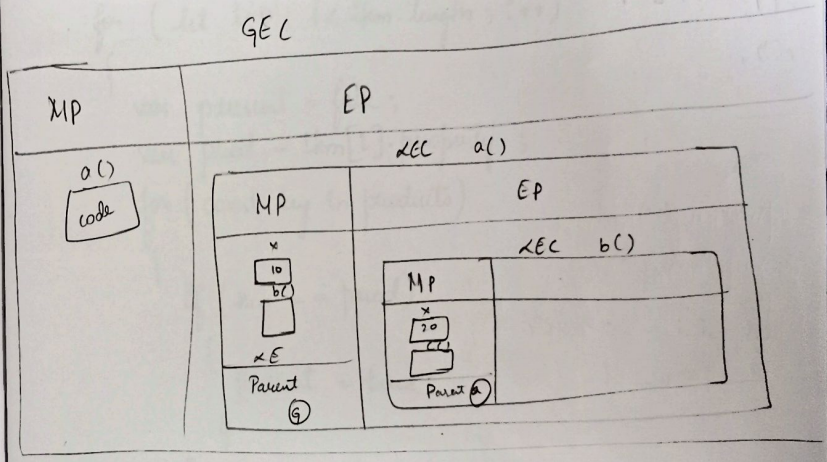
~~Lexical environment~~

```
function a()
{
  var x = 10;
  function b()
  {
    var x = 20;
    function c()
    {
      var x = 30
      console.log(x);
    }
    c();
  }
  b();
}
a();
```

Global Execution Context



→ Lexical environment



CRUD operation

```
var a = 10;
function a()
{
  function b()
  {
    function c()
    {
      console.log(x);
    }
  }
}
a();
b();
c();
```

```
create   var obj = {x:10, y:20}
read     obj.x
update   obj.x = 20;
delete   delete obj.x
```

```
function find (items)
{
  let products = {};

  for (let i=0; i< item.length; i++)
  {
    var present = false;
    var prod = item[i].property;
    for (const key in products)
    {
      if (key == prod)
      {
        present = true;
      }
    }
    if (present == false)
    {
      products[prod] = 1;
    }
    else {
      products[prod] = products[prod]+1;
    }
  }
```

Console.log(products);

Data types

find([{product: "TV", no: 10}
{"FM" no: 20}
{"AC" no: 30}
{"AC" no: 10}
{"TV" no: 50}]

Add 40

60

Section 5

DOM

object CRUD

- Tag + Attributes + Style

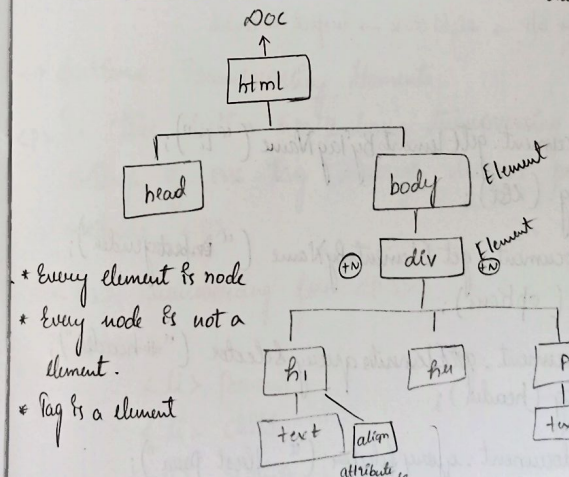
DOM [Document Object Model]

Section 1 :- Selecting elements
Section 2 — Transversing elements } Tags
Section 3 — Manipulation of elements }
Section 4 — Manipulation of attributes } Attributes
Section 5 — Manipulation of Style } Style.
Section 6 — Events

→ DOM tree structure

Node - It is a connection point of one to another node

Nodes
① Normal
② Element → Tag
③ Text → text
④ Attributes → Attribute



* Every element is node
* Every node is not a element.
* Tag is a element

→ by default if we will be there text nodes before & after tags.

object is element attribute nodes

→ Selecting elements by using script tag

We can do in the JS scripting within the html page or external page.

by linking through src = "selecting.js"

— ① Selecting of Tags

① getElementById() → return array → give one Id
② getElementsByClassName() → return in array format
③ getElementsByTagName() } given in array manner
④ getElementsByName()
⑤ querySelector() → give the one Id
⑥ querySelectorAll() → give the array Manner

① Id → It is used identify uniquely.

→ Running → browser → o/p → Right click → Inspect → Console

②

③ let list = document.getElementsByTagName("li");
console.log(list);

④ let options = document.getElementsByName("experiencegender");
console.log(options);

⑤ let header = document.getElementById.querySelector("#header");
console.log(header);

⑥ let para = document.querySelector(".first para");
querySelector("p");

⑥ let para1 = document.querySelectorAll

⑤ querySelector("#id") == getElementById() → returns single element
querySelector(".cls") == getElementByClassName() → return single
querySelector("tag") == getElementByTagName() → return single

① getElementById() → returns single element by Id
② getElementsByClassName → returns multiple elements of class
③ getElementsByTagName → —"— of tag
④ getElementsByName → —"— of name

⑥ querySelectorAll()
querySelectorAll(".cls") → getElementsByClassName → returns multiple elements
querySelectorAll("tag") → getElementsBy

→ Sections: Traversing Elements

 In this section we'll learn traversing of parent, child & siblings of one tag through several properties & methods.

<h3> Traversing for </h3>

 Parent
 Child
 Sibling

onclick = "sample()"
<button> Execute </button>
onhtml click

```
<script>
    let list1 = document.querySelector("li");
      console.log (list1);

  function sample ()
  {
    let list1 = doc.querySelector ("li");
      console.log (list1);
  }
</script>
```

when we don't want to call directly we can use this type

Event listener attributes used for diff tags

① On click → "sample()" we can invoke sample() anywhere in tags
② onDblClick

```
<script>
    function sample()
    {
      let list1 = doc.querySelector("li");
      console.log (list1.parentElement);
      console.log (list1.parentNode);
    }
</script>
```

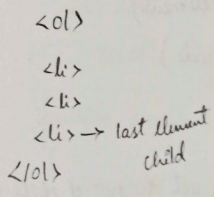
```
    function sample()
    {
      let orderList = doc.querySelector("li");
        console.log (orderList.childElementCount); → 3
        console.log (orderList.childNodes); → 7
    }
```

① Parent
① parentElement → get the parent tag (element)
② parentNode → get the parent tag (node)

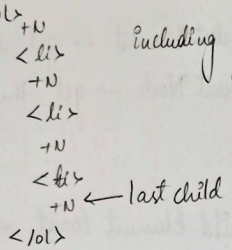
② Child
① Child Element Count → property to get the no. of child tags
② Child Nodes → property to get the no. of child nodes present in parent.
③ Children → gives one element type of child (tag)
④ first Child → it will consider all type of child nodes & gives first child
⑤ first Element Child → it will consider only element nodes & gives first child (tag)

```
<script>
    function sample()
    {
      let orderList = doc.querySelector("ol")
      console.log (orderList.children);
      console.log (document.body.children); // for body we are taking document

    let orderList = doc.querySelector("ol");
      console.log (orderList.firstChild);
      console.log (orderList.firstElementChild);
      console.log (orderList.lastChild);
      console.log (orderList.lastElementChild);
    }
```

first element child



first child




```
<script>
    function sample()
    {
        let order list = doc.query selector("ol");
        console.log(order list.hasChildNodes);
    }
```

⑤ hasChildNodes() → It is a method which returns a boolean if child present or not.

⑥ last child → It will consider all type of nodes & give last child.

⑦ Last Child Element →

```
<script>
    function sample()
    {
        let li = doc.query selector("li");
        console.log(li.nextSibling);
        console.log(li.next Element sibling);
        console.log(li.previous sibling);
        console.log(li.previous Element sibling);
```

③ Sibling :

① Next sibling → consider all nodes and gives next sibling.

② Next Element Nodes → consider only element nodes and give next sibling (tags).

③ previous sibling → consider all nodes & gives previous sibling

④ previous Element sibling → consider only element nodes & gives previous sibling.

Section 3 : Manuplation of Elements

create
Read
update
Delete
} tag through JS

through interface we can change the text

1. Inner Hex

```
<h1> Manuplating the tag </h1>
<h2>
<p> Tags can be manuplated by using several properties
    and methods of DOM </p>
<Input type="text" placeholder="enter the text"></b>
```

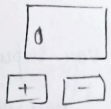
```
<script>
    function sample()
    {
      var head = doc.query selector("h1");
      console.log(head);
    }
</script>
```

o/p <h1> Manuplating the tag <h1>

```
<script>
  function sample()
  {
    var head = doc.querySelector("h1");
    var x = head.innerText;
    console.log(x);
    var x = head.textContent
    console.log(x);
    var x = head.value;
    console.log(x);
  }

function sample()
{
  var head = doc.query selector("input");
  var x = input.textContent;
  console.log(x);
```

Assignment 1

```
<style>
  div{background color:
      width: 250px;
      height: 250px;
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%(left), -50%(top));
    } box sizing: border-box

  input {
      width: 250px;
      height: 200px;
    }

  button {
      height: 45px;
      width: 110px;
    }

</style>
<h1 align="center"> Counter </h1>
<hr>
<div>
    <input type="text" />
    <button click on="increase()"(calculator(+1))> + </button>
    <button> - </button>
</div>  click on = "decrease()"
        calculator(-1)
```



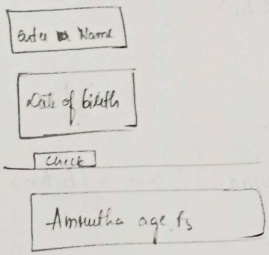
conversion -> javascript will decide

conversion -> we can decide

```
function increase ()
{
    var input = doc.querySelector ("input");
    var val = Number (input.value);
    input.value = value + 1;
}
```

check the combine

```
function decrease ()
{
  var input = document.querySelector ("input");
  var val = Number (input.value);
  input.value = value - 1;
}

function calculate (operator)
{
  var input = document.querySelector ("input");
  var val = Number (input.value);
  input.value = val + operator;
// operator == 1 ? input.value = val + 1 : input.value = val - 1;
```




```
<script>
  function sample ()
  {
    var input = document.querySelector ("input");
    var head = document.CreateElement ("h1");
    head.InnerText = Input value;
    document.body.append (head);
    document.body.prepend (head);
  }
```

Work

```
<h1> food list <h1>
<hr>
<ol>
  <li> Chicken biryani <li>
  <li> Chilly chicken <li>
  <li> lollipop <li>
  <li> Hydrabad c
```

1. Create Element ("Tag")
2. Text Content
3. Inner text
4. value
5. Append (child node) → push acts like push()
6. Prepend (child node) → acts like Unshift()
7. Append Child() → Insert only one element

Section 4

Manipulation of Attributes

① hasAttribute ← Check ("Attribute");
② getAttribute ← Read ("Attribute");
③ setAttribute ← Create, Update, delete ("Attribute", "value");

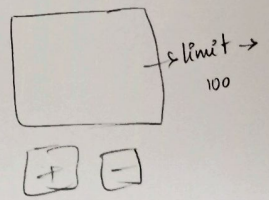
We cannot do delete in Attribute
can do in tag

```
<h1 align = "center"> Manipulation of attributes </h1>
<button onClick = "sample()"> click </button>

<script>
    function sample()
    {
        var head = document.querySelector("h1");
        alert(head.hasAttribute("bgcolor"));
        alert(head.getAttribute("bgcolor"));
        document.body.setAttribute("bgcolor", "white");
```

Section 5

Style

① GetComputedStyle()
② Style



0 - 30 → Green
30 - 50 → Blue
>50 → Red