॥ श्री ॥

Page No.
Date

Variables and Data Types :—

Q) What is Javascript ?

datatype
it calculates at runtime
we can or cannot mention variable data types.

- programming language. Dynamically typed language
- we it to give instruction to computer.

Input (code) → computer → output
instruction.

```
alert ("Apna college");
```
— notification pop-up.

console.log is used to print a message to console.

quotes
single or double. both can be used

```
console.log ("Aniket");
```

How to include in HTML ?

```
<body>
    <script src = "file_name.js"> </script>
</body>
```

Variables :—

- containers to store data.
- eg.

assignment operator.

```
fullName = "chavan";
console.log (fullName);
```
chavan

```
x = null;
```
कोई value नही है

```
y = undefined;
```
पता ही नही इसके अंदर क्या है

camal case :— word का 1st letter small & 2nd word का 1st letter capital.

-e.g. fullName

Page No.
Date

- Rules :—

(i) Variable names are case sensitive ;
i.e. "a" & "A" are different.

(ii) only letters, digits, underscore (_) and $ is allowed (not even space) ✗ full Name = "chavan" ;
space

(iii) only letter, underscore (_) or $ should be 1st character.

(iv) Reserved words can not be variable names.
(like console (small letter में लिखा)

defining variables :—

(i) Var :—
e.g. var age = 24
var age = 25
— variable can be re-declared & updated.
— A global scope variable.

we use mostly ✓ (ii) let :—
— variable can not be re-declared but can be updated.
— A block scope variable.
✓ let age = 24
✗ let age = 25 — no redeclare
age = 26 ✓
let name = "Tony" ;

(iii) const :—
— variable can not be redeclared or updated.
— A block scope variable.

const a ; ✗ error
const a = 10 ; ✓ const variable को शुरुवात में कोई न कोई value देनी है।

typeof → gives datatype. of variable

object के अंदर की change कर सकते है

object const है तो . eg. const

let → changed

const → ✗

Data Types :—

(Number, String, Boolean, undefined, Null, BigInt, Symbol)

↑ all these

(a) Primitive Data types (7)
- fixed, pre-defined
- such as

let x = BigInt("123");

let y = Symbol("Hello!");

(b) Non-primitive :— (objects)
- derived from primitive

arrays, function, dates

(a) objects :— collection of values
- used to represent real-world objects or abstract concepts.
- key-value pairs. {key: value};

string ↵ ↳ can be of any data type

```
let obj = { name : "John";
            age : 30 };

obj.age = 31;  → now age = 31
console.log(obj["age"]);
```

also used to access key

↳ can access key

```
console.log(obj.age);
```

Operators and Conditional Statements :—

(a) Comments :—
- part of code which is not executed.

// for single line comment

/* */ → for multiple line comment.

a ⊕ b → operand

↓
operator

Page No.
Date

Operators :—

- used to perform some operation on data.

(i) Arithematic operators :—

(+, -, *, /, modulus (%), Exponentiation, (**)
increment, decrement)

unary operator → (++) (--)

a**b = a^b

a** 2 = a^2

```
let a=5;
let b=2;
console.log("a+b=", a+b);
                ↳ a+b = 7
console.log(a**b);
                ↳ 5^2 = 25
```

(ii) Unary operator :—

(++) (--)

a++ — post increment (पहले काम फिर value change)

++a — pre increment (पहले value change then काम)

```
console.log(++a); → 6
console.log(a++); → 5   but a = 6 now.
                          console.log(a) → 6
```

(iii) Assignment operator :—

(=, +=, -=, *=, %=, **=)

(iv) comparison operator :—

(Equal to (==), Equal to & type (===), Not equal to (!=), Not equal to & type (!==), >, >=, <, <=)

type: string

"3" == 3 ; → true (only values are compared)

"3" === 3 ; → false (both value & type is compared)

(v) Logical operator :-

(&& , || , ! (NOT))

conditional statements :-

- to implement some condition in the code.

(i) if statement :-

```
let color;
if ( mode == "dark-mode") {
    color = "black";
}
```

(ii) if-else statement :-

```
let color;
if (mode == "dark-mode") {
    color= "black";
}
else {
    color = "white";
}
```

(iii) else-if statement :-

```
if ( age < 18) {
    console.log ( "jr.");
}
else if ( age > 60) {
    console.log ( "senior");
}
else {
    console.log ("middle");
}
```

Alert → one time pop-up
prompt → it can take input.

Page No.
Date

Ternary operator :- if-else in one line

condition ? true output : false output

- eg. let age = 20;
age > 18 ? "adult" : "not adult";

output :- adult (as age > 18 → true)

Loops in JS :-
- used to execute a piece of code again & again.

(i) For loop :- initializing, stopping condition, updation.

```
for (let i=1; i<5; i++) {
    console.log ("Aniket");
}
```

↳ print 5 times Aniket.

(ii) While loop :- initialize - loop ची आधी ↳ termination

```
while (condition) {
    // do some work;
    → updation ही done in loop.
}
```

(iii) do-while loop :- run atleast once.

```
do {
    // do some work
} while (condition);
```

for array & string

(a) For-of loop :- used to iterate over the values of ~~of object~~ (array, string).

```
for (let val of strVar) {
    // do some work
}
```

Page No.
Date

```
let str = "Apna college";
```

↳ initialization, updation, stopping automatic

```
for ( let i of str) {
        console.log (i);
}
```

output :- A
p
n
a
c

(b) for-in loop :- for objects or properties of object (iterate on)

```
for ( let key in objvar) {
        //do some work;
}
```

- e.g.

```
let student = {
        name : "Rahul";
        age : 20,
        cgpa : 7.5,
        isPass : true
};

for ( let kayin student) {

console.log ( key) ;
```

output : name
age
cgpa
ispass.

```
console.log ("key = ", key, "value = ", student[key]);
}
```

value access करण्यासाठी -

Strings :—

— sequence of character wed to represent text.

— 'Hello' or "Hello"

① create string :—

```
let str = "Apna college";
```

② string length :—

```
str.length
```

③ string indices :—

```
str[0], str[1]
```

- Template literals in JS :— (` `)

— a way to have embedded expressions in strings.

```
`this is template literal`
```

```
let sentence = `This is template literal`;
```

e.g.

```
let obj = {
        item : "pen",
        price : 20
    };
```

```
console.log ( " the cost of ", item obj.item, " is", obj.price);
```

↳ The cost of pen is 20.

simple way :-

```
let output = ` The cost of ${obj.item} is ${obj.price}`;

console.log ( output);
```

- **String interpolation** :— we can insert variable / expression directly in template literal.
  — to create strings by doing substitution of placeholders.

  `string text ${ expression / variable } string text`

  string interpolation

- **Escape** characters :— (\n) (\t)
  — used for nextline — used as tab space.

```
let str = "Apna\t college";
console.log( str, " ", str.length);
                ↓          ↓
            Apna college   12  → do not count escape characters
```

String Methods :—
— built-in fun°. to manipulate strings.

(i) str.toUppercase() & str.tolowercase()
— used to convert strings to upper & lower case.

```
let str = "Apnacollege";
str.toUppercase();  → APNACOLLEGE
str.tolowercase();  → apnacollege.

console.log (str); → Apnacollege → original string as any method not make changes in original strings but written new value always.
```

BUT:-

```
str = str.toUppercase();
console.log (str);
```

now changed.

↳ APNACOLLEGE

(ii) str.trim() :—
— removes whitespaces. (starting or ending whitespaces)

will be deleted after trim

let str = " ___ Apna college JS ";
space

console.log (str.trim());

↳ Apna college JS

(iii) str.slice (start , end)

— extract portion of string

not inclusive

i.e. 0 1 2 3
a b c d

slice(1, 3)

↳ bc (1→2)

(iv) str1.concat (str2)

— used to concatenate two or more strings.

let str1 = "Apna";
let str2 = "college";

let result = str1.concat (str2);
console.log (result);

↳ Apnacollege.

OR let res = "hello" + 123;

let res = str1 + str2

↳ also used as concatenat.

(v) str.replace (searchVal, newVal) :-

— used to replace specific character or substring in string.

let str = "hellolll";
console.log (str.replace ("h", "H"));

↳ Hellolll

console.log (str.replaceAll ("ll" "p"));

↳ Hepop.

(vi) str.charAt (idx) :—

— used to retrieve character from given index.

let str = " IloveJS";
console.log (str.charAt(3)); → v

str[0] = "S" → no change in original string

~~let newStr = str[0]~~

```
str = str.replace("I", "i");
```
→ ✓

Arrays :— ↗ object datatype

arrays are mutable

- collection of items.

(i) Create Array :—

```
let heroes = ["ironman", "hulk", "thor"];
let marks = [96, 75, 48, 33];
let info = ["rahul", 86, "delhi"];
```

```
console.log(heroes.length);
```

Looping in Array :— ← for, for of, for in } these preferred mostly.

Q: print all elements of array ?

```
for( int i=0; i<arr.length; i++){
    console.log(arr[i]);
}
```

```
for( let hero of heroes){
    console.log(hero);
}
```

```
hero.toUpperCase(); → IRONMAN
                      HULK
                      THOR
```

Array Methods :—

(i) Push ()—

- add to end.

(ii) pop () : —

- delete from end & return

e.g. let items = ["potato", "apple", "tomato"];

items.push ("chips");

↳ ["potato", "apple", "tomato", "chips"];

items.pop();

↳ ["potato", "apple", "tomato"]

(ii) toString () : — ← नया string written करता है.

— converts array to string.

let marks = [87, 68, 33];

marks.toString();

↳ '87, 68, 33' single string.

(iv) Concat () : —

-joins multiple array & return result

let name = ["rahul"];

let surName = ["x"];

name.concat(surName);

↳ name = ["rahul", "x"];

concat (surName, phone);
↳ multiple

Page No.
Date

(v) unshift () :-
— add to start.

(vi) shift () :-
— delete from start & written.

(vii) slice () :-
— returns piece of array

```
slice ( startIdx , endIdx );
```

(viii) splice () :- → makes changes in original array.
— change original array (add, remove, replace);

```
splice ( startIdx , delCount , newEl1 ... )
```
(new elements add करायचे)
किती items को delete करना है

exclusive
(1) 4 से end तक.
console.log (heroes . slice (1, 3));
↳ 1, 2 not 3
~~ironman~~, ~~thor~~ hulk, thor

Add :-
let marks = [87, 58, 41, 1, 2, 3]
(index start को delete नहीं)
marks . splice (2, 0, 4);
↳
~~marks = [87, 58, 4, 1, 2, 4]~~
marks = [87, 58, 4, 41, 1, 2]

Delete :-
marks . splice (3, 1);
↳ [87, 58, 41, 2, 3]

Replace :- (1 element delete करेगा)
marks . splice (3, 1, 101);
↳ deleted 1 & added 101.
[87, 58, 41, 101, 2, 3]

splice(4) → 4th index के आगे सारे delete करदो.

splice() → nothing happen.

Page No.
Date

Functions :—

— Block of code that performs a specific task, can be invoked whenever needed.

(i) Function definition :—

```
function functionName() {
    // do some work.
}

function functionName(param1, param2 ....) {
    // do some work.
}
```

(ii) Function call :—

```
functionName();
```

```
function learn(msg) {
}

learn("I love JS");
```

// I love JS.

- eg. :—

```
function sum(x, y) {                    block.
    s = x+y; // local variable → scope in this func only
    return s;
}

let val = sum(3,5);
console.log(val); ⟹ 8
```

Higher order Function :— कुछ fun°. को as a parameter we कर रहे होते है या किसी fun°. को written करते है.

Page No.
Date

Array Functions :—

— compact way of writing a function.

without this nothing printed we have to store it in some variable so const.

← const functionName = (param1, param2 ...) => { // do some work }

const variable

tell us what work should perform on input.

```
const sum = (a, b) => {
    return a+b;
}
```

single line also executed as

```
const printHello = () => console.log("Hello");
```

forEach Loop in Arrays :—

— arr.forEach (callBack Function)

↳ fun°. to execute for each element in array.

→ it passed as an argument to another function.

callBack Function

```
arr.forEach((val) => {
    console.log(val);
})
```

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

callBack fun°. value at each index

```
arr.forEach( function printVal (val) {
    console.log(val);
});
```

but callBack fun°. passed as arrowfunction in forEach.

three used

val
idx
arr

```
arr.forEach ((val) => {
    console.log(val);
})
```

↳ 1,2,3,4,5.

e.g. let num = [67, 52, 39];

let calcSquare = (num) => {
    console.log (num * num);
};

nums.forEach (calcSquare);

Array Methods :—

returns
(i) Map :— written new array ને forEach થી નથી કરી શકતા.

— create new array with the result of some operation The value its callback returns are used to form new array.

arr.map (callBackFnx (value, index, array))

let newArr = arr.map ((val) => {
    return val*2;
})

(ii) Filter :—

— creates a new array of elements that give true for a condition / filter.

-e.g. arr = [1, 2, 3, 4, 5, 6];

all even elements from array.

let newArr = arr.filter ((val) => {
    return val %2 === 0;
})

console.log (newArr); → [2, 4, 6].

[1, 2, 3, 4) input में multiple value
return
like sum, avg, single value

(iii) Reduce :—

- performs some operations & reduces the array to a single value.
- It returns that single value.

-e.g.

```
const array1 = [1,2,3,4];

const initialValue = 0;                              result
const sumWithInitial = array1.reduce ((accumulator,
      currentValue) => accumulator + currentValue,
      (var)
      initialValue,
);

console.log (sumWithInitial);
                   ↳ 10      (1+2+3+4)
```

Q) Largest element :—

```
let arr = [5, 6, 2, 1, 3];

const output = arr.reduce ((prev, curr) => {

      return prev > curr ? prev : curr;
});
      → 6
```

DOM :— Document Object Model.

<style> → connects HTML with CSS.
<script> → connects HTML with JS.

(i) ~~Win~~ - data representation of objects that comprise the structure and content of document on the web.
- allow us to create change or remove element from document.

if <script> included before <body> DOM elements are not accessible in it

Page No.
Date

alert
confirm
prompt

- Browser Object Model :-
  - represents additional objects provided by the browsers (host environment) for working with everything expect the document

(i) alert :-
  - used to invoke mini windows (popup) with a msg.

```
alert("Hello");
```

(ii) Prompt :-
  - used to take user input as a string

```
inp = prompt("Hi");
```

(iii) confirm :-
  - shows a msg and waits for the user to press or cancel.
  - Returns true for ok & false for cancle.

Window object :-
  - represents an open window in a browser.
  - It is browser's object (not javascript's) & is automatically created by browser.
  - global object with lots of properties & method.

e.g. (i) document property
  ↳ used to access & manipulate HTML elements.

(ii) location
  ↳ provide info about current url.

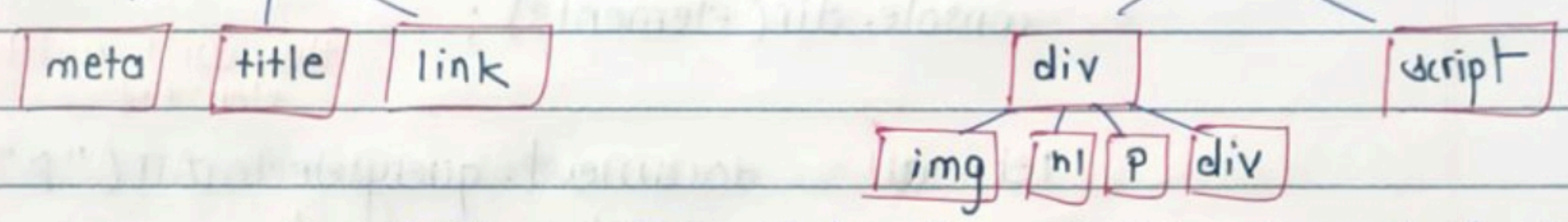
(iii) setTimeout(), setInterval() open(), close()
  scheduling task. open & close windows.

javascript : make changes dynamically at run time

Page No.
Date

Q) What is DOM ?

— when web page is loaded, the browser creates a document object model of the page.



— to access HTML in Javascript. (all HTML elements & tags)

```
console.dir (document.body.childNodes[i]);
```

DOM Manipulation :—

(i) Selecting with id :—

```
document.getElementById ("myId")
```

(ii) Selecting with class :—

```
document.getElementsByClassName ("myclass")
```

(iii) Selecting with tag :—

```
document.getElementsByTagName("p");
```

get → value
set - change

display tostring - representation
↑ console.log
dir ↳ diary
↳ display interactive list of prop

Page No.
Date

Query selector :- returns nodelist - dom tree में हर individual thing is node

(i) document.querySelector ("myId / myclass / tg")
↳ return first element

(ii) document.querySelectorAll ("myId / "myclass" / tg")
↳ retruns Nodelist

e.g.

(i) If I want all paragraphs tags :-

let elements = document.querySelector ("p");
console.dir (elements);

↓
this will written first element

let all = document.querySelectorAll ("p");
console.dir (all);

↓
this will return all elements of paragraph tag..

let firstClass = document.querySelector (".myclass");

↓
dot के साथ लिखते है

firstClass.innerText;

→ for id

Properties :-

(i) tagName :- return tag for element nodes.
(ii) innerText :- return the text content of the element & all its children.
(iii) innerHTML :- returns the plane text or HTML content in the element.
(iv) textContent :- return textual content even for hidden elements.

<h1 style = "visibility: hidden"> old heading </h1>
heading.innerText
→ empty
heading.textContent
→ old heading.

text
comment } node —
element

Page No.
Date

Q: create a H2 heading element with text - "Hello Javascript". Append "from Apna college students" using JS.

```
let h2 = document.querySelector("h2");
console.dir(h2);
```
↳ Hello Javascript

```
h2.innerText = "h2.innerText + " from Apna college";
```
↳ Hello Javascript from Apna college

Attributes :—

(i) getAttribute (attr) :—
— to get attribute value.

(ii) setAttribute (attr, value) :—
— to set attribute val. th

style :—

node.style

```
let para = document.querySelector("p");
console.log(para.getAttribute("class"));
```
↳ return. class name i.e. Para

```
para.setAttribute("class", "newClass");
```

Insert Elements :—

(i) node.append(el) → add at the end of node (inside).
(ii) node.prepend(el) → add at the start of node (inside)

let el = document.createElement("div")

<div → inside
 → outside

(iii) node.before(el) - adds before the node (outside)
(iv) node.after(el) - add after the node (outside.

Delete Element :-

node.remove()

↳ e.g. div.remove

classList :—

- allow for styling the css classes of an element
- read-only property
- includes style & class name.

Q) create <p> tag in html, give it class & some styling. Now create a new class in css and try to append this class to <p> element.

para.classList.add("newclass");

<p class="content newclass" ....>

Events :— used to trigger some operation

- The change in the state of an object
- events are fired to notify code of "interesting changes" that may affect code execution.

(i) Mouse Events - (click, double click, etc)
(ii) Keyboard events - (keypress, keyup, keydown)
(iii) Form events - (submit)
(iv) Print event & many more.

ondblclick = double click

single quotes outside double quotes

```
<button onclick ="console.log('Hello');" < click me </butt>

<div onmouseover = " console.log('Hello'); < this is box</
```

જેટલી જેટલી div પર mouse ફેરવ તેટલી
Hello પ્રિન્ટ console માં.

Event Handling in JS :-

```
node.event = () => {
    //handle here.
}
```

div.~~event~~ ondblclick
btn.onclick

```
let btn = document.querySelector("#btn1");

btn1.onclick = () => {
    console.log("btn click");
    let a = 25;
    a++;
    console.log(a);
}
```

if I have done inline event handling & javascript event handling Javascript code will get priority to run.

Event object :-

- It is a special object that has details about the event.
- All event handlers have access to the Event object's properties & methods.

Inline - code bulky
Jawas - एक बार लिखके एक ही बार use कर सकते है
Listeners - एक event के लिए multiple listners create

```
node. event = (e) => {
    //handle here.
}
```

e.target, e.type, e.clientX, e.clientY.

- e.target ↓ event कहाँ occur हुआ
- e.type ↓ what is type of event
- e.clientX ↓ where the event occur horizontally

```
btn1. onClick = (evt) => {
    console.log(evt);
    console.log(evt.type);      → click
    console.log(evt.target);    → button#btn  (→ id)
};
```

Event Listners :— through this we can add multiple functions to same event (fun).

```
node.addEventListener (event, callback)
node.removeEventListener (event, callback)
```

callback reference should be same to remove

```
btn1. addEventListner ("click", () => {        → object
    console.log(" button1 clicked");
});
```

this in remove

```
~~btn1. add~~
btn1. removeEventListener ("click, () => {
    console.log ("button1 clicked);
});
```

it will not remove as same func. needed

Math.random - 0 - 1 ka

So 8H 0-2 :. math.random() * 3

Page No.
Date

∴ const handler3 = () => {
    console.log ("button1 clicked");
}

btn1. removeEventListener ("click", handler3);
↳ now it will remove

toggle button :- 1st click → dark. } - mode.
2nd click → light.

let modeBtn = document. querySelector ("#mode");
let currMode = "light-mode";
modeBtn. addEventListener ("click", () => {
    // console.log ("you are trying to change mode");

if (currMode == "light-Mode") {
    currMode = "dark";
    document. querySelector("body") . style. background-color = "black";
} else {
    currMode = "light";
    console.log (currMode)
}
});

white = "light";

also

document. querySelector ("body) . classlist . add ("dark");

classes and objects :—

if object & prototype have same method objects method will be used.

(1) Prototypes :—

- Javascript object is an entity having state and behaviour (properties & method)
- JS objects have a special property called prototype.
- we can set it using --proto-- ↳ reference to object.

by using prototype we can use ~~one~~ function to same different object

object method :-

```
const student = {
    fullName : "xyz";
    marks : 94
    printMarks . function () {
        console.log ("marks", this.marks);
    },
};

student . printMarks ();
```

method

we are talking about student object.

this.marks.

→ 94

```
const employee = {
    calTax () {
        console.log ("---");
    },
    calTax2 : function () {
        console.log (---);
    },
};
```

दोनो तरह से define कर सकते है

```
const karan = {
    salary : 5000;
};

karan . __proto__ = employee
karan . calTax ();
```

Page No.
Date

Classes :- जब हमे bulk मे similar object को create करना होता है
blueprint to create n object.
similar

- program code template for creating objects
- those object will have some state (variables) & some behaviour (functions) inside it.

```
class Myclass {
    constructor () { ... }
    myMethod () { ... }
}

let myObj = new Myclass ();
```

this will automatically invoked when we create new object

```
class Toyotacar {
    start () {
        console.log ("start");
    }
    stop () {
        console.log ("stop");
    }
}

let fortuner = new Toyotacar ();
```

object की property

```
setBrand (brand) {
    (this.brand) = brand;
}
```

variable

यहाँ object के लिए func. call होगा।

```
fortuner.setBrand = ("Fortuner");
```

type : object

now start & stop func. is available in fortuner

```
fortuner.start ();
```

constructor :-
- automatically invoked by new
- initializes object.
- we have to do some option at the time initialization

```
constructor ( brand, mileage ) {
    console.log ( "new object" );
    this.brand = brand;
    this.mileage = mileage;
}
```

Inheritance :—

— passing down properties & methods from parent clas to child clas.

```
clas parent {
}
    clas child extends Parent {
    }
```

if child & parent have same method, childs method will wed [method overriding]

Day-11

Selecting by IDs, classes and more :—

i) getElementbyID :—

— part of Javascript document object
— it allows to access specific element on a web page by its unique ID.

html code

```
<div id = "myDiv" > This is my Div </div>
```

JS code :

```
let myDiv = document.getElementById ("myDiv");
myDiv.innerHTML = "This is my new text";
```

• Advantages :-

(i) not need to traverse entire DOM.
(ii) useful when working with large & complex web pages.

— It returns first element that matches specific ID.
— case sensitive.

i.e. myDiv & mydiv

different.

ii) getElementsByClassName :—

— to access multiple elements on web page by their class name.

— returns live HTML collection of elements with given class name or empty HTML collection if no such element found.

html code :

```
<div class = "myClass" > 1st </div>
<div  —11—           >  2nd </div>
<div  —11—           >  3rd </div>
```

Javascript code :-

```
let elements = document.getElementsByClassName
                  ("myClass);
for (let i=0; i<elements.length; i++) {
      elements[i].innerHTML = "New text";
}
```

iii) getElementsByName :-

— access multiple elements by their name attributes.

html code:

```
<input type="text" name="myName" value:"">
<      —11—                              >
<      —11—                              >
```

Javascript code:

```
let elements = document.getElementsByName ("myName"
for (i=0; i<elements.length; i++) {
      elements[i].value = " new value";
}
```

iv) getElementsByTagName :—

— access multiple elements on web pages by their HTML tag name.

html code :

```
<p>  1st para  </p>
<p>  2nd  —11—  </p>
<p>  3rd  —11—  </p>
```

Javascript code :

```
let elements = document.getElementsByTagName("p");
for (i=0; i<elements.length; i++){
    elements[i].innerHTML = "new text";
}
```

- innerHTML :—

— allows to access and manipulate the HTML content of element
— returns content bet. opening & closing tag of an element as a string of HTML
— used to insert any valid HTML code, including scripts & event handler

- outerHTML :—

— access & manipulate entire HTML of an elemet, including element's own tags.
— returns entire HTML of an element as a string

inline css using Javascript :-

```
document.querySelector(".box").style.background color = "red";
```

1st class will be red

all class elements :-

we can not directly use what we done above

```
document.querySelectorAll(".box").forEach(e => {
    e.style.background color = "yellow";
});
```

returns all multiple elements in node list so we have to iterate it & work with their individual elements

we have to use forEach to access all elements.

Inserting and Removing elements using Javascript :-
(innerHTML and outerHTML)

```
document.designmode = "on"
```

यानें आपण कोणती पण वेबसाईट वर changes करू शकतो.

Page No.
Date

Day12 # Events, Event Bubbling, setInterval and setTimeout:-

1) console.dir function :-

- shows element DOM tree
- -11- as an object with its properties.

- tagName / nodeName :-
  - used to read tag name of an element

    i) tagName - only for element nodes
    ii) nodeName - defined for any node. (text, comment)

- setTimeout and setInterval :-

- setTimeout allows us to run a function once after the interval of time.
- syntax:

```
let timerId = setTimeout(function, <delay>, <arg1>,
                         <arg2>)
```

delay in ms

returns timerId.

- clearTimeout :-
  - used to cancel execution (in case we change our mind)
  - eg.

```
let timerId = setTimeout( () => alert("never"),
                          1000);
clearTimeout(timerId);
```

Page No.
Date

- **setInterval :-**

- **syntax :-**

```
let timerId = setInterval ( fun, <delay> , <arg1> , <arg2>]
```

— unlike setTimeout, it runs the function not only once but regularly after the given interval of time.

- **clearInterval :-**

— to stop setInterval.

Browser Events :-

— event is signal that something has happened. All the DOM nodes generate this signal.

— some important DOM events :-

(i) **Mouse Event :-**

— click, context menu (right click), mouseover/ mouseout, mousedown/ mouseup, mousemove

(ii) **Keyboard Events :-**

— keydown & keyups.

(iii) **Form Element Events :-**

— submit, focus etc.

(iv) **Document events :-**

— DOMContentLoaded.

Handling Events :-

— Events can be handled through HTML attribute

<input value = "Hii" onclick = "alert('hey')" type = "button" >

↓

can be another javascript function.

— Events can also handled through the onclick property.

```
elem.onclick = function () {
                    alert("yes")
};
```

Note :- adding handler with Javascript overwrites the existing handler.

addEventListner and removeEventListner :-

— used to assign multiple handlers to an event.

element.addEventListner (event, handler)

element.removeEventListner (event, handler)

↳ handler must be same function object for this to work.

The Event object :—

— when an event happens, the browser creates an event object, puts details into it and passes it to handler as an argument.

```
elem.onclick = function (event) {
  ---
}
```

event.type : Event type
event.currentTarget : element that handled the event
event.clientX / event.clientY : co-ordinates of cursor.

Day-13

Callback Functions :—

— used to call function once a function executes

- Asynchronous nature of JS :—

```
console.log ("I am Aniket")
console.log ("I am Rohan")

setTimeout ( () => {
    console.log ("Hii")
}, 2000);

console.log ("The End");
```

output :-

(i) Expected

I am Aniket
—11— Rohan
Hii
The End

ii) Actual

I am Aniket
—11— Rohan
The End
Hii.

★ — callback function is a function paised to another function as an argument which is then invoked inside the outer fun. to complete actions

e.g.:

```
const callback = (arg, fn) => {
                          console.log(arg)
                          fn()
function loadScript (src, callback) {
  let script = document.createElement ('script')
  script.src = src
  script.onload = () => callback (script)
  document.head.append (script)

— Now we can do something like this :

loadScript ('https://cdn.harray.com', (script) => {
  alert ("script is loaded")
  alert (script.src)
});
```

— Why ? :-

— function को on demand run कराना
— ↓ fn as a variable pass कर सकते
है।

— To manage code in better way we we promises which are

- **Promises :—**

— solution to callback hell (world was callback hell) is promises.

— it is the "promise of code execution". The code either executes or fails in both the cases

— **syntax :—**

```
let promise = new Promise ( function (resolve, reject) {
                    —11—
});
```

— "**resolve & reject**" are two callbacks provided by javascript.

resolve (value) → if job finished successfully
reject (error) → if job fails.

- **Properties :—**

1) State : Initially pending, then changes to either "fulfield" when resolve is called or "rejected" when reject is call.

2) Result : Initially undefined then changes to value if resolved or error when rejected.

Consumers :-

— code can receive the final result of a promise through "then and catch".

```
promise.then (function (result) {
    —II— },
    function (error) {
    —II—
});
```

- If we are inserted only in errors we can we null as first argument or we 'catch"

```
promise.catch (alert);
```

Promise chaining :-

— we can chain promises & make them passed the resolve value to one another.

```
p.then ( func. (result) => {
    alert (result) ;
    return 2;
}); then.
```

- The idea is to pass the results through the chain of .then handlers.