JavaScript

* To print some output on console

```
console.log('Hello');
console.log("World");
console.log(`Hehehe`);
```

You can use '', "", ``.

* Note:- To link JS file in html

```
<script src='learn.js"></script>
```

* To create pop-up window.

```
window.alert("This is a pop-up window with a button. OK at the corner");
```

Note:- If more than one window alerts are given another appears after clicking ok of before one

* Comment

```
// This is comment
/*
  This
  is
  comment
*/
```

* Note:- When you want to print something on page the code should be written in

```
window.onload = function(){ }
```

To print in console no need of it.

(OR) place the <script> link just before the </body>

Because, to keep script tag in head, we should make sure that JS code runs after the DOM is fully loaded.' (DOM-Document Object Model)
↳ Structure of HTML

* To print on webpage

let, in html file:-

```
<body>
    <h1 id="myH1"></h1>
    <p id="myP"></p>
</body>
```

→ Instead of typing here we type here

In JS file:-

```
document.getElementById("myH1").textContent = `Hello`;
"  "  ("myP").  "  = `Paragraph`;
```

Variables

declaration — assignment (No need of specifying datatype any where)

```
let x;
x = 100;
```

(OR)

```
let x = 100;
```

~~This helps in overwriting the value~~

Note:- Declaration will be done only once. Overwritting can be done for multiple times.

* To print variable in a statement:-

```
let x = 18;
console.log('I'm ${x} years old'); //In case of including variable use only `...`
```

* To find datatype:-

```
console.log(typeof x); // Output: number.
```

Note:- For booleans also it is same.

Arithmetic Operators:-

operands (values, variables, etc)
operators (+, -, *, /, %)

* Precedence:-
  (), **, */1/%, +/-

* To take user input:

```
Ex:- let x;
     x = window.prompt("Enter x:"); (Takes input in pop-up window)
     console.log(x); (Takes input in the console-
```

* To take user input on the webpage:-

* In html file:-

```
<input id="myText"> <br>
<button id="mySubmit"> Submit </button>
```

In JS file:-

```
let text;
document.getElementById("mySubmit").onclick = function(){
        text = document.getElementById("myText").value;
        console.log(text);
}
```

Datatype Conversion:-

```
Ex: let x = "pizza";
    let y = "";
    let z = "";
    x = Number(x);
    y = String(y);
    z = Boolean(z);
    console.log(x, typeof x);    // Nan 'number'
    console.log(y, typeof y);    // pizza 'string'
    console.log(z, typeof z);    // true 'boolean'
```

Note:- Prints Nan 'number' (If no value is assigned to the variable or can't convert into datatype)
undefined 'string'
false 'boolean'

* Constant (const) = a variable can't be changed:-
  Declared same as let x; for this
  const x;

Note:- Can be written as:-

```
const
document.getElementById("mySubmit").onclick = function(){}
                    (OR)
const submit = document.getElementById("mySubmit");
submit.onclick = function(){}
```

~~count++;~~
~~countLabel.text~~

* Math :-

Math = built-in object that provides a collection of properties and methods.

Ex:- `console.log(Math.PI);` // prints π value.

* π = Math.PI
* e = Math.E
* Math.round();-> rounds a number to an integer.
* Math.floor();-> rounds down.
* Math.ceil();-> " up
* Math.trunc(); -> eliminates decimal
* Math.pow(x,y); -> x^y
* Math.sqrt(x); -> $\sqrt{x}$
* Math.log(); logarithm (base = e)
* Math.sin(), Math.cos(), Math.tan() } trigonometric
* Math.abs() -> turns -ve to +ve (absolute value)
* Math.sign() -> For -ve -> -1
  " 0 -> 0
  " +ve -> 1
* ~~Math.max()~~ -> Math.max(x,y,z) -> displays max among
  Math.min(x,y,z) -> " min "

Math.random() -> prints random no.,

Ex:- let x = Math.random(); (Generates random no: b/w 0 & 1)

~~console.log(randomNum~~

console.log(x);

Note :- location.reload(); -> Refreshes the page.

* If Statements = if a condition is true, execute some code if not, do something else

Ex:-
```
let age;
age=25;                          -> Condition; You can even write booleans in the conditions.
if (age >= 18) {
    console.log("You are permitted");
}else{
    console.log("You should grow up man");
}
```

* Nested if statements :- If in If statement.
* else if : can be used for more statements/conditions

Ex:-
```
let x = 18;
if (x<18){
    console.log("less than 18");
}else if (x=18){
    console.log("It is 18");
}else{
    console.log(">18");
}
```

* .checked -> property that determines the checked state of an HTML checkbox or radio button element.

Note :- for attribute is also same as Id.

Ex:-
```
<body>
<input type="checkbox" id="myCheckBox">
<label for="myCheckBox">subscribe</label><br>

<input type="radio" id="visaBtn" name="card">        (This name should be same for all radio buttons (It makes sure that user can select only one))
<label for="visaBtn">Visa</label><br>
<!-- Do the same radio buttons for Master Card & PayPal -->
<button type="submit" id="mySubmit">submit</button>
```

```
<p id="subResult"></p>
<p id="paymentResult"></p>
```

(JavaScript)

JS code:-

```
const myCheckBox = document.getElementById("myCheckedBox");
//Do the same for every Id mentioned in html.
mySubmit.onclick = function(){
    if(myCheckBox.checked){
        subresult.textContent= `You are subscribed!`;
    }
    else{
        subresult.textContent=`You are not subscribed`;
    }
    if(visaBtn.checked){
        paymentResult.textContent=`You are paying with
                                   Visa`;
    }else if(masterCardBtn.checked){
        paymentResult.textContent=`You are paying with
                                   Mastercard`;
    }else if(payPalBtn.checked){
        paymentResult.textContent=`You are paying with PayPal`;
    }else{
        paymentResult.textContent=`You must select a payment
                                   type`;
    }
}
```

* Ternary Operator:-

```
let age = 21;
let message = age >= 18 ? "You're adult" : "You're a minor";
                          Prints if true     Prints if false
console.log(message);
```

* SWITCH:-

```
let day = 1;
switch(day){
  case 1:
    console.log("It's Monday");
    break;
  case 2;
  // continues till case 7 (Sunday)
  default:
    console.log(`${day} is not a day`);
}
```

* String Methods:-

```
let x = "JS";
console.log(x.charAt(0)); // Prints "J"
console.log(x.indexOf("S")); // Prints "1"
console.log(x.length); //Prints 2
let y= "pujibabu";
console.log(y.lastIndexOf("b")); // Prints the index of 1st "b" from
                                    last i.e., "6"
letz = "  puji  ";
console.log(z.trim()); // Removes extra spaces (prints: "puji")
console.log(z.toUpperCase()); // Prints "PUJI".
console.log(z.toLowerCase()); //  "  "  "puji"
```

```
console.log(z.repeat(2)); // Repeats 2 times i.e. prints
                          //"pujipuji"
console.log(z.startsWith("p"); //Prints true
  "   "   "      "  ("P"); //  "   false
console.log(z.includes(" "); //  "   false
  "   "   "   "  ("u"); //  "   true
let num="79896-25748";
console.log(num.replaceAll("-",""); // 79896257 48
console.log(num.padStart(15,"0"); // 000079896-25748
  "   "  (  "  .padEnd(15,"0"); // 79896-257480000
```

* String slicing :- creating a substring from a portion of another string

string.slice(start,end)

```
const name = "Puji Babu";
let x = name.slice(0,4); //Puji
let y = name.slice(5,9); //Babu
```

* When can use -ve index even.

```
let firstname = name.slice(0, name.indexOf(" ")); //Puji
let lastname = " " + (name.indexOf(" ")+1) //Babu
```

↳ End is not needed it prints till last index

* Method chaining :- Calling one method after another in one continuous line of code.

Ex:-

```
//No Method changing
username = username.trim();
let letter = username.charAt(0);
letter = letter.toUpperCase();
let extrachars = username.slice(1);
extrachars = extrachars.toLowerCase();
username = letter + extrachars
console.log(username);
```

```
//Method Changing
username = username.trim().charAt(0).toUpperCase() + username.trim().
           slice(1).toLowerCase();
console.log(username);
```

* Logical operators :- used to combine or manipulate Boolean values (true or false)

AND - &&
OR - ||
NOT - !

Mostly used in if,else, for loops, etc...

Some more operators :-

= assignment op
== comparison op (equal)
=== strictly equality op (compare if values & datatype are equal)
!= Not equal (Inequality op)
!== strictly not equal (Strictly Inequality op)

Ex-1 :-

```
const x = 3;
if(x=="3"){
  console.log("That is true");
}else{
  console.log("No");
}//Prints That is true
if(x==="3"){
    console.log("Yes");
}else{
    Console.log("NO");
// Prints ("No")
if(x===3){
    console.log("Yes");
}else{ console.log("No");
// prints "Yes".
```

* In the same way "!==" operator works checks both datatype along with value.

While Loop

```
let name = "";
while (name === ""){
    name = window.prompt(`Enter your name`);
}
console.log(`Hello ${name}`);
```

Do While Loop

```
let username;
do{
username = window.prompt(`Enter your name`);
}while(username === "" || username == null)
console.log(`Hello ${username}`);
```

FOR LOOP

```
for (let i=0; i<=2; i++){
    console.log(i);
}
```

(initialization; continuation (condition); increment or decrement)

Note:- isNaN(variable name)
This indicates the variable is not of datatype.

* Function ():- (parameters)

```
function hbd(name, age){
    console.log(`Happy birthday Mr. ${name}`);
    console.log(`You're now ${age} years old`);
}
hbd(Pujith, 18); //Calling the function
```

(Arguments)

Returning output from function:-

```
function add(x,y){
    let result = x+y;
    return result; -> if it is not returned it won't print the answer
}
let answer = add(2,3);
console.log(answer);
```

* Local variable vs Global Variable.

Variables declared inside the func (valid only for that func)

Variable declared outside the function and is valid everywhere

Importance: Local > Global

* Array

```
let fruits = ["apple", "orange", "banana"];
```

1) Indexing
```
fruits[3] = "coconut";
```
2) length
```
console.log(fruits.length);
```
3) fruits.push("coconut"); (Adds at end)

4) fruits.pop(); (Removes last one)

5) fruits.unshift("mango"); (Adds at beginning)

6) fruits.shift(); (Removes 1st one)

7) Finding index
```
let index = fruits.indexOf("apple");
console.log(index);
```

* Spread operator :- ... (unpacks strings and array).

Ex: let numbers = [1,2,3,4,5];
```
let maximum = Math.max(...numbers);
console.log(maximum);
```
If code is run without (...) o/p = NaN

Ex: let username = "Bro Code";
```
let letters = [...username];
console.log(letters);
```
o/p -> ['B','r','o',' ','C','o','d','e']

Same but inline-2

Ex: let letters = [...username].join("-");

o/p -> B-r-o- -C-o-d-e

Ex: let odds = ["1", "3"];
```
let evens = ["2", "4"];
let nums = [...odds, ...evens];
console.log(nums);
```

* Rest parameters (...rest)

Bundles seperate elements into an array.

Ex: function openFridge(...foods) {
```
    console.log(foods);
}
const food1 = "pizza";
const food2 = "hamburger";
const food3 = "hotdog";
const food4 = "sushi";
const food5 = "ramen";

openFridge(food1, food2, food3, food4, food5);
```
~~o/p -> ['pizza', 'hamburger', 'hotdog', 'sushi', 'ramen']~~

o/p -> pizza, hamburger hotdog sushi ramen.

In this place if we write
```
~~const foods~~
function getFood(...foods) {
    return foods;
}
const foods = getFood(food1, food2, ..., food5);
console.log(foods);
```

* Call Back :- a func that is passed as an argument to another func

```
Ex: hello(bye);          → parameter
function hello(callback){
    console.log("Hello!");
    callback();                    O/P :- Hello!
}                                         Bye!
function bye();
    console.log("Bye!");
}
```

* forEach() :- method used to iterate over the elements of an array and apply a specified function (callback) to each element

Note: array.forEach(callback)

```
Ex: let numbers = [1,2,3,4,5];     //array
    numbers.forEach(display);      //array.forEach(callback)
    function(display)
    function display(element){
        console.log(element);
    }

Ex: let numbers = [1,2,3]
    numbers.forEach(square);
    function square(element, index, array){
        array[index] = Math.pow(element,2);
    }
    numbers.forEach(display);
    function display(element){
        console.log(element);
    }
```

* .map - accepts callback and applies that function to each element of an array, then return a new array.

.map is a mini version of forEach which implements forEach and maps to the variable itself with just map keyword.

```
Ex: const numbers = [1,2,3,4,5];
    const squares = numbers.map(square);
    console.log(squares)                    O/P :- [1,4,9,16,25]

    function square(element){
        return Math.pow(element,2);
    }
```

* .filter = creates a new array by filtering out elements.

```
Ex: let numbers = [1,2,3,4,5,6,7]
    let oddNums = numbers.filter(isOdd);
    console.log(oddNums);
                                            O/P :- [1,3,5,7]
    function isOdd(element){
        return element % 2 !== 0;
    }
```

* .reduce = reduce the elements of an array to a single value.

```
Ex: const prices = [5,30,10,25,15,20]
    const total = prices.reduce(sum);
    console.log(`$${total.toFixed(2)}`);        O/P :- 105.00
    function sum(accumulator, element){
        return accumulator+element;
    }
```

Note: first value will be assigned to accumulator and it goes on changing due to operations with element. Here element is next data of accumulator.

* function :-

Declaration :-
Define a reusable block of code that performs a specific task

Ex :- function hello(){
    console.log("Hello");
}

expressions
a way to define functions as values or variables.

(●●) Ex :- const hello = function(){
    console.log("Hello");
}

Ex :- const numbers = [1,2,3]
const squares = numbers.map(square);
console.log(squares);
function square(element){
    return Math.pow(element,2);
}

Ex :- const numbers = [1,2,3];
const squares = numbers.map(
function(element){
    return Math.pow(element);
});
console.log(squares);

* arrow functions :- concise way to write function expressions.
Note :- (parameters) => some code in func

1) Ex :- const hello = () => console.log("Hello"); // instead of (●●) Ex in function eg above
hello();

Note :- setTimeout() function
Ex :- setTimeout(() => console.log("Hello"), 3000);
It prints/displays "Hello" after 3 secs.

2) Ex :- For map :-
const squares = numbers.map((element) => Math.pow(element,2));
console.log(squares);

* Object :- A collection of related properties and/or methods. Can represent real world objects (people, products, etc).
object = { key : value, function() }

Ex :- const person = {
    Name : "Pujith",
    age : 30,
    isEmployed : true
}
console.log(person.Name);
console.log(person.age);
console.log(person.isEmployed);

Ex :- const person = {
    Name : "Pujith",
    sayHello : function(){
        console.log("Hi! I am Pujith")
    },
}
console.log(person.Name);
person.sayHello();

* this :- reference to obj person.name = this.name :-
Ex :- const person = {
    name : "Pujith",
    sayHello : function() {console.log(`Hi! I am ${this.name}`)},
}
person.sayHello();

Note :- "this" keyword doesn't work with arrow functions.

* constructor :- special method for defining the properties & methods of objects
Ex :- function Car(model, color){ → parameters
    this.model = model
    this.color = color
} } constructor

~~// constructor~~ :
const car1 = new Car("Mustang","red"); → obj Name, → Arguments, } constructor
console.log(car1.model);
console.log(car1.color);

Classes :- Provides more structured and cleaner way to work with objs compared to traditional constructor functions
ex: like:- encapsulation, inheritance, static keyword.

```
class Product {
    constructor(name, price) {
        this.name = name;
        this.price = price;
    }
    display_Product() {
        console.log(`Product: ${this.name}`);
        console.log(`Price: $${this.price.toFixed(2)}`);
    }
    calculateTotal(salesTax) {
        return this.price + (this.price * salesTax);
    }
}
const salesTax = 0.05;
const product1 = new Product("Shirt", 1999);
const product2 = new Product("Pants", 22.50);
const product3 = new Product("Underwear", 100.00);

product1.displayProduct(); //calling func1
const total = product1.calculateTotal(salesTax); //calling func2 $ assigning value to total
console.log(total);
```

constructor
func1
func2
class
objs

* Static : A keyword that defines properties or methods that belong to a class itself rather than the objects created from that class.

Ex:
```
class Mat {
    static PI = 3.14;                    // static variable

    static getDiameter(radius) {        // static method
        return radius * 2
    }
}
console.log(Mat.PI)
console.log(Mat.getDiameter(10));
```

* Inheritence :-

Ex:
```
class Animal {
    alive = true;
    eat() {
        console.log(`This ${this.name} is eating`);
    }
}
class Rabbit extends Animal {
    name = "rabbit";
}
class Fish extends Animal {
    name = "fish";
}
const rabit = new Rabbit();
const fish = new Fish();
console.log
fish.eat();
rabit.eat();
```

* super(superclass) :- Used in classes to call the constructor or access the properties & methods of a parent ~~(superclass)~~

this : this object

super : the parent

Note : when 2 child classes share similar parameters super key word must be used

Ex :-
```
class Animal {
    constructor(name, age){
        this.name = name;
        this.age = age;
    }
}
class Rabbit extends Animal {
    constructor(name, age, runspeed){
        super(name, age);
        this.runspeed = runspeed;
    }
}
class ~~Rabit~~ fish extends Animal {
    constructor(name, age, swimspeed){
        super(name, age); }
        this.swimspeed = swimspeed;
    }
}
```

In place of
```
super();
this.name = name;
this.age = age;
```

* getter = special method that makes a property readable.
setter = " " " " " " writable.

Ex :-
```
class Rectangle {
    constructor(width, height){
        this.width = width;
        this.height = height;
    }
    set width(newWidth){
        if (newWidth > 0){
            this._width = newWidth;
        }
        else{
            console.error("Width must be positive");
        }
    } get width(){return this._width;} get area(){ return(this._width*this._hei
}
const rectangle = new Rectangle(-100, 20);
console.log(rectangle.width);
```

* destructuring :- Extract values from arrays & objects, then assign them to variables in a convenient way.

[] => to perform array destructuring.

{} => " " object " .

swap the value of two variables
```
let a = 1;
let b = 2;
[a,b] = [b,a];
console.log(a);      // 2
console.log(b);      // 1
```

Ex-2 :-

```
const person1 = {
    fName : "spongebob",
    lName : "squarepants",
    age:30,
    job:"Fry Cook",
}
const  person2 = {
    fName : "Patrick",
    lName : "star",
    age : 34,
}

const {fName, lName, age, job="Unemployed"} = person2;

console.log(fName);
console.log(lName);
console.log(age);
console.log(job); //As in the structure there is no job mentioned
                  //by default it prints unemployed.
```

parameter → fName, lName, age, job="Unemployed"
argument → person2

→ in display function

You can use functions to display like

* Nested objects :- Objects inside other objects.

```
Ex: const person = {
    Name: {fName:"Pujith", lName:"Sri Sai"},
    age : 18,
    isStudent:true,
    hobbies: {"Karate","cooking"},
}
console.log(person.hobbies[0]); //Karate
console.log(person.Name.fName); // Pujith
```


* To display nested object :-

```
for(const property in person.address){
    console.log(person.address[property]);
}
```

Note :- `const fruits = [{name:"apple", color:"Red"},`
`{ " ":"Orange", color:"Orange"}];`
`console.log(fruits[0].name);`

* Sort method :- sort()

```
let fruits = [1,2,3,5,6,9,8,4]
fruits.sort();
console.log(fruits);
```

Ex: `fruits.sort((a,b)=> b-a);`

Note: We can sort objects in order by name, age, etc

Ex:- In place of `a → a.age & b → b.age`

* Fisher - Yates algorithm:

* Date object :- `// Date(year, month, day, hour, minute, second, ms)`

```
const date = new Date();
console.log(date);
```

Note:- Some imp codes

```
const year = date.getFullYear();
" month = " . " Month();
" day = " . getDate();
" hour = " .getHours();
" min = " . getMinutes;
" sec = " .getSeconds
" dayofWeek = const . get Day();
```

To set :-

date.setFullYear(2024); // Same for remaining

* Closure: A function defined inside of another function
the inner func has access to the variables
and scope of the outer function.
Allow for private variables & state maintenance
Used frequently in JS frameworks: React, Vue, Angular.

Ex: function outer() {
let message = "Hello";
function Inner() {
console.log(message);
}
inner();
}
message = "Goodbye";
outer;

O/p :- Hello

* setTimeOut() :- setTimeout(callback, delay);
clearTimeOut :- (timeoutId) = can cancel a timeout before it triggers

Ex: const timeoutId = setTimeout(() => window.alert("Hello"), 3000);
clear(timeoutId);

* Some Imp :-
Ex: const hours = now.getHours().toString().padStart(2, 0);
→ pads beginning with 0
→ If 1 digit then 1st digit will be 0.
To place 2 digits always

Similarly :- padEnd(2,0) will replace 0 with last digit.
pads ending with the 0.

• setInterval(updateClock, 1000);

Note :- In css for color codes we can use hsl codes like
h -> Hue (Number)
s -> Saturation (%)
l -> Lightness (%)
Even hsla (a)
a -> alpha (0.0 - 1)
(-> transparency

In css flexdirection is one keyword in flex for display (can be column or row)

* ES6 Module :-
An external file that contains reusable code that can be imported into other Javascript files. Write reusable code for many diff apps.

* Synchronous :- Executes line by line consecutively
Asynchronous :- Allows multiple operations to be performed concurrently without waiting
Doesn't block execution flow & allows the program to continue
Handled with : Callbacks, Promises, Async/wait

* Error: A object is created to represent the problem in code.

To overcome we use
try{} = Encloses code that might potentially cause an error.
catch{} = Catch & Handle any thrown Errors from try{}
finally{} = (optional) Always executes. Used mostly for clean up
ex. close files, close connections, release resources

Ex: try {
console.log("Hello"); // Error
}
catch(error) {
console.log(error);
} finally { console.log("This always executes"); } → This is also printed
console.log("You have reached"); // this will be printed.

Document Object Model :- (DOM)

Object{} that represents the page you see in web browser.
& provides you API to interact with it.
Web browser constructs the DOM when it loads an HTML document
& structures all the elements in a tree like structure all the elements in a tree-like representation.
Javascript can access the DOM to dynamically change the content, structure & style of a webpage.

Note :- console.log(document); // Prints html code
console.dir(document); // Prints properties of document.

```
document.title = "My Website";
document.body.style.backgroundColor = "hsl(0, 0%, 15%)";
```
↳ CSS is possible in JS

* Element selectors :-

1. document.getElementById(); // Element OR NULL
2. " .getElementsClassName(); // HTML collection
3. " .getElementByTagName(); // HTML collection
4. document.querySelector(); // ~~Element OR NULL~~ First Element OR NULL
5. " .querySelectorAll(); // NODELIST

Note :- To select single element in HTML collection of class use indexes.
Ex: fruits.style.backgroundColor = "yellow";

~~We~~ can also use for()

Note: Foreach() is not applicable on HTML collection
So,
Array.from(fruits).forEach();
convert into array
& then apply forEach();
for loop can be used directly

Ex. for querySelector :-

```
const fruits = document.querySelector(".fruits"); // Can access only first element
```

querySelectorAll :- Similar to HTML collection (Named Nodelist)

```
const foods = document.querySelectorAll("li");
```

Note :- For this (Nodelist) forEach(method) can be used directly

* DOM Navigation :-

- firstElementChild
- lastElementChild
- nextElementSibling
- previousElementSibling
- parentElement
- children

Ex :-
```
const element = document.getElementById("fruits");
const ~~first Element~~ first = element.firstElementChild;
first.style.backgroundColor = "yellow";
```

Manipulating HTML Elements using JS :-

Steps : 1). CREATE the Element.
2) ADD Attributes / Properties
3) APPEND or PREPEND Element to DOM.
4) REMOVE HTML element.

Append-add at end
Prepend - add to begin
Append() : in the given

* eventListener: listen for spectic events to create interactive webpages
events: click, mouseover, mouseout
.addEventListener (event, callback);

Ex: const myBox = document.getElementById ("myBox");

function changeColor(event) {
    event.target.style.backgroundColor = "tomato";
}
myBox.addEventListener("click", changeColor);
event ↓ ↓ callback function.

* Some more imp events -> Keydown (pressdown the key) & Keyup (releasing the key)

• document.addEventListener("keydown", event => {
    console.log(event);
});

* const myBox = document.getElementById("myBox");
const moveAmount = 10;
let x = 0;
let y = 0;

document.addEventListener("keydown", event => {
    if (event.key.startsWith("Arrow")) { event.preventDefault(); }
    switch (event.key) {
        case "ArrowUp":
            y -= moveAmount;
            break;
        case "ArrowDown":
            y += moveAmount;
            break;
        case "ArrowLeft":
            x -= moveAmount;
            break;
        case "ArrowRight":
            x += moveAmount;
            break;
    } myBox.style.top = `${y}px`; myBox.style.left = `${x}px`;
});

Note:- To hide an element:-
Elename.style.display = "none";
To show the hidden element:-
Elename.style.display = "block";

If we use visibility in place of display when element is even hidden the place will not be occupied by other.

Nodelist: static collection of HTML elements by (id, class, element)
can be created by using querySelectorAll()
Similar to an array, but no (map, filter, reduce) [Nodelist {}]
Nodelist won't upd

Ex: let buttons = document.querySelectorAll(".myButtons");
buttons.forEach(button => {
    button.style.backgroundColor = "green";
    button.textContent += "O";
});

* Adding an element using JS

```
const newButton = document.createElement("button");
newButton.textContent = "Button 5";
newButton.classList = "myButtons";
document.body.appendChild(newButton);
console.log(buttons);
// But this new br
// To show buttons in console
buttons = document.querySelectorAll(".myButtons");
console.log(buttons);
```

To remove element → event.target.remove();

* classList = Element property in JS used to interact with an element's list of classes (css classes)
Allows you to make reusable classes for many elements across your webpage.
add(), remove(), toggle(Remove if present, Add if not), replace(Old class, new class), contains().

Ex: `const myButton = document.getElementById("myButton");`
`myButton.classList.add("enabled");`
↳ It will add the element to class. ↳ Then it will have the css properties of enabled class.

Note: Arguments should be given in single quotes in HTML

* Callback Hell:- Situation in JS where callbacks are nested within other callbacks to the degree where the code is difficult to read. Old pattern to handle asynchronous functions.
Use Promises + async/await to avoid Callback Hell.

Functions should be written as follows:-

```
function task(callback){
    setTimeout(() => {
        console.log("Task completed");
        callback();
    }, 3000);
```

Ex:
For These:-

```
task();
task1(() => {
    task2(() => {
        task3(() => {
            task4(() => console.log("All tasks completed"));
        })
    });
});
```

* Promise:- An Object that manages asynchronous operations.
Wrap a Promise object around {asynchronous code}.
"I promise to return a value".
PENDING → RESOLVED or REJECTED
`new Promise((resolve, reject) => {asynchronous code})`

Ex:
* Do these chores in order
Walk the dog
Clean the kitchen
Take out the trash.

Instead of of callback Hell we use method chaining which is Promise

Ex:
```
function walkDog(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("You walk the dog");
        }, 1500);
    });
}
function cleanKitchen(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolves("You clean the kitchen");
        }, 2500);
    });
}
function takeOutTrash(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("You take out the trash");
        }, 500);
    });
}
walkDog().then(value => {console.log(value); return cleanKitchen()})
    .then(value => {console.log(value); return takeOutTrash()})
    .then(value => {console.log(value); console.log("You finished all chores")});
```

The use of reject :-
```
function walkDog(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const dogWalked = true;
            if (dogWalked){
                resolve("You walk the dog");
            } else{
                reject("You didn't walk the dog");
            }
        }, 1500);});
```

* Async/Await :- Async = makes a func return a promise
  Await = makes an async func wait for a promise.

- Allows you write asynchronous code in a synchronous manner
- Async doesn't have resolve or reject parameters
- Everything after Await is placed in an event queue.

Ex:
```
async function do_chores(){
    try{
        const walkDogResult = await(walkDog);
        console.log(walkDogResult);

        const cleanKitchenResult = await cleanKitchen();
        console.log(cleanKitchenResult);

        const takeOutTrashResult = await takeOutTrash();
        console.log(takeOutTrashResult);

        console.log("You finished all chores");
    }catch(error){
        console.error(error);
    }
}
do_chores();
```

in this place we write

These funcs should be written in synchronous way.

*JSON: (JS Object Notation) data-interchange format
Used for exchanging data b/w a server & a web application.
JSON files {key:value} OR [value1, value2, value3].

- JSON.stringify() = converts a JS object to a JSON string.
- JSON.parse() = converts a JSON string to a JS object.

Ex: const names = ["Puji", "Pujith", "PujiBabu"];
const jsonString = JSON.stringify(names);
console.log (jsonString); // prints list as a string [—,—,—]
console.log (names); // prints list and set of strings independently.

Ex: const jsonNames = ["Puji", "Pujith", "PujiBabu"];
const parsedData = JSON.parse (jsonNames);
console.log (jsonNames); //Prints JSON obj as [0—, 1—, 2—]
↳ Prints string representation of array.

console.log (parsedData); //Prints as JS obj

*To fetch the data from JSON file we use fetch() method.

Ex: fetch ("RelativePath.json")
.then(response => response.json())
.then (values => values.forEach(value => console.log(value.name)))

//Prints each element of the array in data

Note: You can have a data as 1 object, Group of objects (with {key:value} pairs) or simply an array.

*fetch = Func used for making HTTP req's to fetch resources.
(JSON style data, images, files)
Simplifies asynchronous data fetching in JS & used for interacting with API's to retrieve & send data asynchronously over the web.
fetch (url, {options})
↳ by default this will be GET

Ex: fetch(" ")
.then (response => console.log (response))
.then (data => console.log (data)) → if it is data.id then gives only the value of key "id"
.catch (error -> console.log (error));

Ex: fetch(" ".)
.then response => {
if (!response.ok){
throw new Error("Fetching Failed!");
} return response.json();
})
.then (data => console.log (data.id))
.catch(error => console.error(error);

You can also use async func:-
fetchData();
async function fetchData(){
try{
const response = await fetch(" ")
if (!response.ok){
throw new Error ("could not fetch resource");
}
const data = await response.json();
console.log (data);
} catch(error) {console.error(error);}