**Functions :**

* Function decleration vs expressions

// function decleration

  function walk () {

    console.log('walk');

  }

// anonymous function expression

 const run = function () {

      console.log('run');

 };

// calling with other name

let another = run ;

run ();

another();

* Hoisting

In javaScript Hoisting is a process of moving function declaration to the top of the file . and this is done automatically by the javascript .

* Arguments

function sum (a,b) {

   let total = 0;

   for (let value of arguments)

    total += value;

return total;

}

console.log(sum (1,2,5,10));

* The rest operator

  A type of parameter that gets all of the remaining parameters of a function call via an Array.

function sum (discount , ...prices) {

    const total = prices.reduce((a,b) => a+b);

    return total \* (1-discount);

}

console.log(sum(0.1,20,30,1));

* Default parameters

function intrust (principal , rate = 3.5 , years = 5) {

 return  principal \* rate / 100 \* years ;

}

console.log (intrust(100000));

* Getters and setters

Getters and setters are functions that allow you to get and set object values, respectively. Getter functions return the value of the specified property, while setter functions can set the value of the specified property.

const person = {

    firstname : 'mosh',

    lastname : 'hamadani',

    get fullname () {

        return ` ${person.firstname}  ${person.lastname}`

    },

    set fullname (value) {

        const parts = value.split(' ');

        this.firstname = parts[0];

        this.lastname = parts[1];

    }

}

person.fullname = 'elsa khan';

console.log(person);

* Try and catch

const person = {

    firstname : 'mosh',

    lastname : 'hamadani',

    get fullname () {

        return ` ${person.firstname}  ${person.lastname}`

    },

    set fullname (value) {

        if (typeof value !== 'string')

        throw new Error ('value is not a string');

        const parts = value.split(' ');

        if (parts.length !==2)

        throw new Error (' enter first and last name ')

        this.firstname = parts[0];

        this.lastname = parts[1];

    }

}

try {

person.fullname = 'elsa khan';

}

catch (err){

    alert(err);

}

console.log(person);

* Local vs Global Scope

const color = 'red';

  function start() {

    const color  = 'white'; // value inside a funciton has high weight as compared to global value

    console.log(color);

  }

 function stop () {

    const color = 'green';

    console.log(color);

 }

  start();

  stop();

* Let vs var

function start () {

    for (var i = 0 ; i < 10 ; i++)

    {

            console.log(i)

    }

        console.log(i);

}

start();

//  this is the issue with var

* This keyword

The **object** that is executing the current function.

Method => object

Function => global (window, global)

// Method => object

// Function =>  global (window, global)

const video = {

 title : 'a',

Tags : ['a', 'b' , 'c'],

showTags () {

    this. Tags .forEach (function (Tags){

          console.log(this.title, Tags);

    },this)

}

};

video.showTags();

* Changing this

// 1st solution  use arrow function

const video = {

    title : 'a',

   Tags : ['a', 'b' , 'c'],

   showTags () {

       this. Tags .forEach ( Tags => {

             console.log(this.title, Tags);

       })

   }

   };

//    video.showTags();

//    second solution use methods like bind , call , apply

// example

 function playvideo (a,b) {

console.log (this);

 }

 playvideo.call ({name: 'ali'}, 1,2);

 playvideo.apply ({name: 'asad'},[1,2]);

 const fn = playvideo.bind ({name: 'saad'}, 1,2)();

 playvideo();

* Exercise 1 sum of arguments