**Javascript Notes**

**Function Constructor**

const myFunction = new Function("a", "b", "return a \* b");

myFunction(4,3)

**Function Hoisting**

myFunction(5)

function myFunction(x){

return x\*x

}

**Self InvokingFunction**

(function(){

let x= “Hello”

})();

**Function Argument count**

function myFunction(a,b,c){

return arguments.length; // here you can get number of propertics length like 3

}

myFunction(4,5,6)

**Argument Object**

function findMax() {

    let max = 0;  //-2345789561247867

    for(let i = 0; i < arguments.length; i++) {

      if (arguments[i] > max) {

        max = arguments[i];

      }

    }

    return max;

  }

  console.log(findMax(4,5,6))

**What is this?**

In JavaScript, the **this** keyword refers to an **object**.

this is not a variable. It is a keyword. You cannot change the value of this

Methods like call(), apply(), and bind() can refer this to any object.

**Function Call**

const person = {

    fullName: function() {

      return this.firstName + " " + this.lastName;

    }

}

  const person1 = {

    firstName:"John",

    lastName: "Doe"

  }

  const person2 = {

    firstName:"Mary",

    lastName: "Doe"

  }

  // This will return "John Doe":

  person.fullName.call(person1);

//   another call function

const person3 = {

    fullName: function(city, country) {

      return this.firstName + " " + this.lastName + "," + city + "," + country;

    }

  }

  const person4 = {

    firstName:"John",

    lastName: "Doe"

  }

  const z= person3.fullName.call(person4, "Oslo", "Norway");

console.log(z)

**Function Apply**

// Function Apply

const person6 = {

    fullName: function() {

      return this.firstName + " " + this.lastName;

    }

  }

  const person7 = {

    firstName: "Mary",

    lastName: "Doe"

  }

  // This will return "Mary Doe":

  const aa= person6.fullName.apply(person7);

  console.log(aa);

## The Difference Between call() and apply()

The difference is:

The call() method takes arguments **separately**.

The apply() method takes arguments as an **array**.

//Apply Method in diff

const person8 = {

    fullName: function(city, country) {

      return this.firstName + " " + this.lastName + "," + city + "," + country;

    }

  }

  const person9 = {

    firstName:"John",

    lastName: "Doe"

  }

  console.log(person8.fullName.apply(person9, ["Oslo", "Norway"]));

**Function Bind**

// Bind

const person12 = {

    firstName:"John",

    lastName: "Doe",

    display: function () {

      let x = document.getElementById("demo");

      x.innerHTML = this.firstName + " " + this.lastName;

    }

  }

  let display = person12.display.bind(person12);

  setTimeout(display, 3000);

**Class Methods**

Use the keyword **class** to create class

Always add a **constructor()** method

class Car{

constructor(name, year){

this.name=name;

this.year=year;

}

age(){

const date=new Date();

return date.getFullYear()-this.year;

}

}

const myCar= new Car(“Ford”,2014);

console.log(myCar.age());

**Class Inheritance**

Use **extend** keyword to create inheritance

The **super** method refers to thyen parent class. Bycalling

class Car {  
  constructor(brand) {  
    this.\_carname = brand;  
  }  
  get carname() {  
    return this.\_carname;  
  }  
  set carname(x) {  
    this.\_carname = x;  
  }  
}  
  
const myCar = new Car("Ford");  
myCar.carname = "Volvo";  
document.getElementById("demo").innerHTML = myCar.carname;

**Callback**

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

 function myDisplayer(some) {

    console.log(some);

  }

  function myCalculator(num1, num2, myCallback) {

    let sum = num1 + num2;

    myCallback(sum);

  }

  myCalculator(5, 5, myDisplayer);

Right: myCalculator(5, 5, myDisplayer);

Wrong: ;

**Asynchrous**

  // Asynchrous

  setTimeout(myFunction, 3000);

  function myFunction() {

    console.log("I love You !!");

  }

**Promise**

A Promise is a javascript object that link producing code and consuming code

Promises object properties

* Pending => undefined
* Fuilfiled => a result value
* Rejected => an error object

**Async Await**

async function getFile() {

    let myPromise = new Promise(function(resolve) {

      let req = new XMLHttpRequest();

      req.open('GET', "mycar.html");

      req.onload = function() {

        if (req.status == 200) {

          resolve(req.response);

        } else {

          resolve("File not Found");

        }

      };

      req.send();

    });

    let res= await myPromise;

    console.log(res);

  }

  getFile();

let promise = new Promise(function (resolve, reject) {

setTimeout(function () {

resolve('Promise resolved')}, 4000);

});

// async function

async function asyncFunc() {

// wait until the promise resolves

let result = await promise;

console.log(result);

console.log('hello');

}

// calling the async function

asyncFunc();

**Local Storage in JS**

// Storing data:

const myObj = {name: "John", age: 31, city: "New York"};

const myJSON = JSON.stringify(myObj);

localStorage.setItem("testJSON", myJSON);

// Retrieving data:

let text16 = localStorage.getItem("testJSON");

let obj = JSON.parse(text16);

document.getElementById("demo1").innerHTML = obj.name;

**Sending Data to Server**

const myObj = {name: "John", age: 31, city: "New York"};  
const myJSON = JSON.stringify(myObj);  
window.location = "demo\_json.php?x=" + myJSON;

**Reciving Data from Server**

const myJSON = '{"name":"John", "age":31, "city":"New York"}';  
const myObj = JSON.parse(myJSON);  
document.getElementById("demo").innerHTML = myObj.name;

**JSON Server**

const xmlhttp = new XMLHttpRequest();  
xmlhttp.onload = function() {  
  const myObj = JSON.parse(this.responseText);  
  document.getElementById("demo").innerHTML = myObj.name;  
};  
xmlhttp.open("GET", "json\_demo.txt");  
xmlhttp.send();

async function loadImages(){

  const response=await fetch('https://jsonplaceholder.typicode.com/todos/1');

  const data1=await response.json();

  console.log(data1)

}

loadImages()