# Types of function

//function statement

function a(){

  console.log("a called");

}

//function expression

let b = function(){

  console.log("b called");

}

//anonymous function

function(){

}//will cause an error

//name function expression

let x = function xyz(){

}

//the difference between function statement and function expression is hoisting function expression behaves like a variable and if you call it before initialize it will be undefined

* Call, apply and bind are the types of function call
* Through them we can manage and reuse a function as well as data

let obj = {

  name:"adil",

  father: "younas"

}

let obj2 = {

  name:"rehman",

  father:"tahir"

}

let obj3 = {

  name:"Sheikh",

  father:"abdullah"

}

function getTime(x,y,z){

  console.log(`my name is ${this.name} ${this.father} ${x} and ${y} ${z}`);

}

///////////////////

getTime.call(obj3,"x","y","z")

////////////////

let array = ["x","y","z"]

getTime.apply(obj2,array)

/////////////

let x = getTime.bind(obj,"x","y","z")

x()

# Closure

* Functions together with their and their parent’s lexical environment call closure
* Bundle of scoops are called closure
* Provides security

    function a(a,b,c){

      // let a = 1

      return {

        getTime1:function(){

          console.log(a+b);

        },

        getTime2:function(){

          console.log(a+b+c);

        }

      }

    }

    let store = a(1,2,3)

    store.getTime1()

    store.getTime2()

    let store2 = a(4,5,6)

    store2.getTime1()

    store2.getTime2()

function x(){

  for(var i = 0; i<=5; i++){

    setTimeout(function(){

      console.log(i);

    },i \* 1000)

  }

}

x()

# Call back function

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

//call back function

const perOne = (name, callback)=>{

  console.log(`i am busy right now ${name}`);

  callback()

}

const perTwo = ()=>{

  console.log(`i am perTwo`);

}

perOne("adil",perTwo)

why we use it because of this we call perOne but perTwo call first and vice versa

//why use

const perOne = (name)=>{

  setTimeout(()=>{

    console.log(`i am busy right now ${name}`);

  },3000)

}

const perTwo = ()=>{

  console.log(`i am perTwo`);

}

perOne("adil")

perTwo()

# Why we use promise and asyn await

//promise and async await is use for data that incoming from somewhere and will take time if you rapidly log it, it will show pending data so we use them

function x(){

  let response = fetch("https://jsonplaceholder.typicode.com/users")

  console.log(response); //show pending stage

}

x()

//or

function t(){

  let response = fetch("https://jsonplaceholder.typicode.com/users")

  return response

}

console.log(t());//show pending stage

//if we are using promise method then we use resolve and reject and pass the waiting value to it and bypass it by then

const prom = new Promise((resolve,reject)=>{

  let response = fetch("https://jsonplaceholder.typicode.com/users")

  resolve(response)

})

prom.then((response)=>{

  console.log(response);

})

//if we are using async await then there are two type

//type one deal return

let asn = async function(){

  let response = await fetch("https://jsonplaceholder.typicode.com/users")

  return response

}

asn().then((response)=>{

   console.log(response);

 })

//type 2 deal without return

let asn2 = async function(){

  let response = await fetch("https://jsonplaceholder.typicode.com/users")

  console.log(response);

}

asn2()

# Promise (needs resolve and reject)

* If value came it will store in resolve and reject
* If you don’t use resolve and reject then you are unable to return something
* We use promise because we want to that do that and then run resolve and reject
* For the sake of accessing data, you must write resolve and reject
* To avoid that we use async await

let pro = new Promise((resolve,reject)=>{

  let error = false

  if (!error) {

    // console.log("pass");

    let obj = {

      name:"adil"

    }

    resolve(obj)

  } else {

    console.log("failed");

    reject()

  }

})

pro.then((res)=>{

console.log(res);

})

# Async Await (need return for bypass using then only, otherwise it works)

* Give promise
* If values are came then it will store before await variable
* Resolve and reject are not necessary but return is according to desire

//if we are using async await then there are two type

//type one deal return

let asn = async function(){

  let response = await fetch("https://jsonplaceholder.typicode.com/users")

  return response

}

asn().then((response)=>{

   console.log(response);

 })

//type 2 deal without return

let asn2 = async function(){

  let response = await fetch("https://jsonplaceholder.typicode.com/users")

  console.log(response);

}

asn2()