**Synchronous function**

----------------------

console.log("test")

console.log("test")

console.log("test")

**ASynchronous function**

----------------------------

console.log("test")

console.log("test")

console.log("test")

setTimeout(()=>{

console.log("ram");},1000

)

console.log("test")

console.log("test")

console.log("test")

**callback function**

a callback is a function passed as an argunment to another function.

this technique allows a function to call another function

a callback function can run after another function has finished

**user defined call back function**

**Example 1**

function fun1(x,cbf)

{

Console.log(“function 1”)

Console.log(x)

Cbf();

}

Function fun2()

{

Console.log(“function 2”)

}

Fun1(“hello”,fun2) //

**Example 2**

function showMessage(callback) {

console.log("in side")

setTimeout(() => {

console.log("This is first line");

callback(); // will be called when the task will be finished

}, 3000);

}

showMessage(() => console.log("This is second line"));

//predefined asyn function

setTimeout(callback, 2000); //Async function call through setTimeout() method

function callback() {

console.log("Callback function called");

}

console.log("After function call");

**Promise**

A promise is a TypeScript object which is used to write asynchronous programs. A promise is always a better choice when it comes to managing multiple asynchronous operations, error handling and better code readability.

**How to Create Promise?**

A TypeScript promise takes inner function, and that inner function accepts resolve and rejects parameter

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

// code goes here

});

**Understand TypeScript Promise Parameter:**

A promise accepts callback function as a parameter.

Callback function accepts 2 parameter resolve and reject.

If condition is true then it returns resolve else it returns the reject.

**What are Promise States:**

**State Description**

pending This state refers to the first state when the promise is neither fulfilled nor rejected.

fulfilled As the name suggested when the promise operation executed successfully.

rejected This state refers to the state when the promise operation is failed.

**Example –1**

<script>

var p1=new Promise(function(x,y)

{

var res =false;

if(res)

{

x("it is true");}

else

{y("it is false");}

})

console.log(p1)

p1.then(function(val) {console.log(val);} )

.catch(function (val) {console.log(val);} )

.finally("test")

</script>

**Example - 2**

function asyncAction() {

var promise = new Promise((resolve, reject) => {

setTimeout(() => {

console.log("Async is done!");

resolve("test");

}, 1500);

});

return promise;

}

asyncAction().then(

() => console.log("Resolved!")

);

Error

function asyncAction() {

var promise = new Promise((resolve, reject) => {

setTimeout(() => {

console.log("Async is done!");

reject('Rejected!');

}, 1500);

});

return promise;

}

asyncAction().then(function(success) {

console.log(success);

})

.catch(function(error) {

// error handler is called

console.log(error);

});

**Promise methods**

Promise.all()— it displays the all promise results

Promise.all([p1,p2,p3,p4]) .then(values)=>console.log(values)

Promise.allSettled() // all promises

Promise.any() // it displays true promises

Promise.race() // it prints the first executed promise

**chaining of promises**

**step1(10,false)**

**.then((result1)=>step2(result1,false)**

**.then((result2)=>step3(result2,false)**

**.then((result3)=>console.log(result3)**

**.then((error)=>console.log(error))**

**Async/Await**

are make promises easier to write

Async=>makes a function return a promise

await=>makes a function wait for a promise

function step1(value,error)

{

return new Promise((resolve,reject)=>

{

if(!error)

{

resolve(value+10)

}

else

{

reject("error");

}

});

}

async function result()

{

let result1=step1(10,false);

console.log(result1)

}

result();