Callback

- Function passed as argument in another function

Example 1 - How simple callback is!

```
function myFunc1(callback) {
  console.log("Function is doing task 1 ");
  callback();}

myFunc2(() => {
  console.log("function is doing task 2");});

myFun1(myFunc2); // myFunc2 is callback
```

Output

```
function is doing task 1
function is doing task 2
```

Example 2 - Callback prog in diff ways

- Way 1

```
function getTwoNumbersAndAdd(number1, number2, onSuccess, onFailure) {
  if (typeof number1 === "number" && typeof number2 === "number") {
    onSuccess(number1, number2);
  } else {
    onFailure();
  }
}

function addTwoNumbers(num1, num2) {
  console.log(num1 + num2);
}

function onFail(){
    console.log("Wrong data type");
    console.log("please pass numbers only")
}
// Using already made functions while calling
getTwoNumbersAndAdd(4, 4,addTwoNumbers, onFail);
```

- Way 2

```
function getTwoNumbersAndAdd(number1, number2, onSuccess, onFailure) {
  if (typeof number1 === "number" && typeof number2 === "number") {
    onSuccess(number1, number2);
  } else {
    onFailure();
  }
}

// Directly writing func defination while calling
getTwoNumbersAndAdd(4, 4,
(num1, num2) => {
  console.log(num1 + num2);},
() => {
    console.log("Wrong data type");
    console.log("please pass numbers only")}
);
```

Callback Hell

Task

- Text Delay Color one 1s Violet two 2s purple three 2s red four 1s Pink five 2s green six 3s blue seven 1s brown (heading 1 ke 1sec baad heading change ho, so everytime reference is previous heading - order is imp)

```
const heading1 = document.querySelector(".heading1");
.
.
const heading5 = document.querySelector(".heading5");

setTimeout(()=>{
    heading1.textContent = "one";
    heading1.style.color = "violet";
        setTimeout(()=>{
            heading2.textContent = "two";
            heading2.style.color = "purple";
                setTimeout(()=>{
                    heading3.textContent = "three";
                    heading3.style.color = "red";
                        setTimeout(()=>{
                            heading4.textContent = "four";
                            heading4.style.color = "pink";
                                setTimeout(()=>{
                                    heading5.textContent = "five";
                                    heading5.style.color = "green";
                            },2000)
                        },1000)
                    },2000)
                },2000)
},1000)
```

Pyramid of Doom

```
function changeText(element, text, color, time, onSuccessCallback,
onFailureCallback) {
    setTimeout(()=>{
        if(element){
            element.textContent = text;
            element.style.color = color;
            if(onSuccessCallback){onSuccessCallback();}}
        else{
            if(onFailureCallback){onFailureCallback();}}}
    ,time)}
```

- Pyramid of Doom

```
changeText(heading1, "one","violet",1000,()=>{
  changeText(heading2, "two","purple",2000,()=>{
    changeText(heading3, "three","red",1000,()=>{
      changeText(heading4, "four","pink",1000,()=>{
        changeText(heading5, "five","green",2000,()=>{
          changeText(heading6, "six","blue",1000,()=>{
            changeText(heading7, "seven","brown",1000,()=>{
              changeText(heading8, "eight","cyan",1000,()=>{
                changeText(heading9, "nine","#cda562",1000,()=>{
                  changeText(heading10, "ten","dca652",1000,()=>{

                  },()=>{console.log("Heading10 does not exist")})
                },()=>{console.log("Heading9 does not exist")})
              },()=>{console.log("Heading8 does not exist")})
            },()=>{console.log("Heading7 does not exist")})
          },()=>{console.log("Heading6 does not exist")})
        },()=>{console.log("Heading5 does not exist")})
      },()=>{console.log("Heading4 does not exist")})
    },()=>{console.log("Heading3 does not exist")})
  },()=>{console.log("Heading2 does not exist")})
},()=>{console.log("Heading1 does not exist")})
```

- ['coffee', 'chips','vegetables','salt','rice']

- Promise - fried rice
- fulfill
- reject

Promise

```
const bucket = ['coffee', 'chips','vegetables','salt','rice'];

// Produce Promise
const friedRicePromise = new Promise((resolve,reject)=>{
    if(bucket.includes("vegetables")&& bucket.includes("salt") &&
bucket.includes("rice")){
        resolve({value:"friedrice"});
    }else{
        reject("could not do it");}
})

// Consume Promise
friedRicePromise
.then(  // jab promise resolve hoga
    (myfriedRice)=>{console.log("lets eat ", myfriedRice);})
.catch(  // jab promise reject hoga
    (error)=>{console.log(error)})
```

Simplify Promise Code

Produce Promise

```
const promiseName = new Promise((resolve, reject) =>{
    if(...){
        resolve(pass something)
    }else{
        reject(pass error)}
})
```

Consume Promise

```
promiseName
.then((passed thing)=>{....})
.catch((error)=>{....})
```

Working of Promise with setTimeout and sync code

```
// Promise
1. console.log("script start");
2. const bucket = ['coffee', 'chips','vegetables','salt','rice'];
3. const friedRicePromise = new Promise((resolve,reject)=>{...............})
4. friedRicePromise
   .then()
   .catch()
5. setTimeout(()=>{
    console.log("hello from settimeout")},0)
6. for(let i = 0; i <=100; i++){
    console.log(Math.random(), i);}
7. console.log("script end!!!!")
```

Output

```
script start
100 0.55638752853
script end
promise code
setTimeout code
```

---

1. script start prints
2. variable stored in memory of Global Execution Context
3. Promise produced as Promise object in memory of GEC
4. `Promise is consumed by Browser` -> Promise code goes to browser, it goes in -> `Microtask Queue`-> then/ catch
5. `setTimeout also goes to browser` -> after that time interval, it goes in -> `Callback Queue`
6. 100 Random numbers are printed
7. script end prints -> GEC is cleared
8. promise code prints
9. setTimeout code prints

Priority of Promise > SetTimeout

Priority of Microtask Queue > Callback Queue

Function returning Promise

```
function ricePromise(){
  const bucket = ['coffee', 'chips','vegetables','salts','rice'];

  return new Promise((resolve,reject)=>{
    if(bucket.includes("vegetables")&& bucket.includes("salt") &&
bucket.includes("rice")){
        resolve({value:"friedrice"});
    }else{
        reject("could not do it");
    }})}
```

- Use Promise

```
ricePromise() //promise name nhi function calling
.then(
  (myfriedRice)=>{console.log("lets eat ", myfriedRice);})
.catch(
  (error)=>{console.log(error)})
```

- Resolve/ Reject Promise after 2 seconds

```
function myPromise(){
    return new Promise((resolve, reject)=>{
        const value = true;
        setTimeout(()=>{
            if(value){resolve();
            }else{reject();}
        },2000)
    })}

myPromise()
    .then(()=>{console.log("resolved")})
    .catch(()=>{console.log("rejected")})
```

Promise.resolve

- It `takes value`
- `Returns a promise that resolves with that value`

```
const myPromise = Promise.resolve(5); //Returns promise that resolves with value 5
myPromise.then(value=>{ // value = 5
  console.log(value); /// 5
})
```

- `return value` ~ `return Promise.resolve(value)` -> value is a promise

then()

- `Always returns a promise`
- Therefore, we can create Promise Chaining
- If we den't return -> `return undefined;` happens internally

Promise Chaining

```
// create promise function
function myPromise(){
  return new Promise((resolve, reject)=>{
    resolve("foo");
  })
}

myPromise()
  .then((value)=>{ // value is the value given when promise resolved
    console.log(value); /// foo
    value += "bar";
    return value // value is a promise*** ~ return Promise.resolve(value)
  })
  .then((value) =>{
    console.log(value); /// foobar
    value += " store";
    return value; // value is a promise***
  })
  .then(value=>{
    console.log(value); /// foobar store
  })
```

Callback hell to Flat Code-> Using Promise

```
const heading1 = document.querySelector(".heading1");
const heading10 = document.querySelector(".heading10");

function changeText(element, text, color, time) {
    return new Promise((resolve, reject) => {
        setTimeout(()=>{
            if(element){
              element.textContent = text;
              element.style.color = color;
              resolve();
            }else{
              reject("element not found");}
          },time)
    })}
changeText(heading1, "one", "red", 1000)
  .then(()=>{
    return changeText(heading2, "two", "purple", 1000)}) // returned for next then
  .then(()=>{
     return changeText(heading3, "three", "green", 1000)})
  .catch((error)=>{
      alert(error);
  })
// or
changeText(heading1, "one", "red", 1000)
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading3, "three", "green", 1000))
  .then(()=>changeText(heading10, "ten", "orange", 1000))
  .catch((error)=>{
      alert(error);
  })
```