Class-7: Promises

Callback function

- let's write a function for eatBreakfast

```
function eatBreakfast(item){
    console.log("I will eat"+" "+item +" "+ "as my breakfas
t")
}

eatBreakfast("idly")

Output : I will eat idly as my breakfast
```

- Now let's to pass number as argument along with string

```
function eatBreakfast(item,time){
    console.log("I will eat"+" "+item +" "+ "as my breakfas
t"+"at"+" "+time)
}

eatBreakfast("idly",9)

Output : I will eat idly as my breakfast at 9
```

- Now let's try to pass functions as argument along with strings and numbers

```
function eatBreakfast(item,time){

    console.log("I will eat"+" "+item +" "+ "as my breakfas
t"+"at"+" "+time)
}
```

```
function doBrush(){
    console.log("First brush your teeth")
}

eatBreakfast("idly",9,doBrush)

Output : I will eat idly as my breakfast at 9
```

- We have passed function as argument but how to access callback function

```
function eatBreakfast(item,time,doBrush){
    doBrush()
    console.log("I will eat"+" "+item +" "+ "as my breakfas
t"+"at"+" "+time)
}

function doBrush(){
    console.log("First brush your teeth")
}

eatBreakfast("idly",9,doBrush)

Output :
First brush your teeth
I will eat idly as my breakfast at 9
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-s
    <title>Document</title>
  </head>
  <body></body>
</html>
```

```
<script>
  function eatBreakfast(item, time, doBrush, drinkWater) {
    doBrush();
    console.log(
      "I will eat" + " " + item + " " + "as my breakfast" + "at
    );
    drinkWater();
  }

  function doBrush() {
    console.log("First brush your teeth");
  }

  function drinkWater() {
    console.log("Drink water");
  }

  eatBreakfast("idly", 9, doBrush, drinkWater);
</script>
```

Basic example for washing clothes to understand callback hell

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
    <title>Document</title>
  </head>
  <body></body>
</html>
```

```
<script>
  //callbacks
  function washClothes(time) {
    setTimeout(() => {
      console.log("Clothes are washed!");
    }, time); // Washing takes 2 seconds
  }

  function dryClothes(time) {
    setTimeout(() => {
      console.log("Clothes are dry!");
    }, time); // Drying takes 3 seconds
  }

  function foldClothes(time) {
    setTimeout(() => {
      console.log("Clothes are folded!");
    }, time);
  }

  washClothes(10000);
  dryClothes(10000);
  foldClothes(10000);
</script>
```

adding callback

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-s
    <title>Document</title>
  </head>
  <body>
```

```
    <script>
      // Define the functions with callbacks
      function washClothes(time, callback) {
        setTimeout(() => {
          console.log("Clothes are washed!");
          callback();
        }, time); // Washing takes time specified in millisecon
      }

      function dryClothes(time) {
        setTimeout(() => {
          console.log("Clothes are dry!");
        }, time); // Drying takes time specified in milliseconds
      }

      function foldClothes(time) {
        setTimeout(() => {
          console.log("Clothes are folded!");
        }, time); // Folding takes time specified in millisecon
      }

      // Sequentially process clothes by passing callback functi
      washClothes(2000, () => {
        dryClothes(3000);
      });
    </script>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-s
```

```
    <title>Document</title>
  </head>
  <body>
    <script>
      function washClothes(time, callback) {
        setTimeout(() => {
          console.log("Clothes are washed!");
          callback();
        }, time); // Washing takes time specified in millisecon
      }

      function dryClothes(time, callback) {
        setTimeout(() => {
          console.log("Clothes are dry!");
          callback();
        }, time); // Drying takes time specified in milliseconds
      }

      function foldClothes(time) {
        setTimeout(() => {
          console.log("Clothes are folded!");
        }, time); // Folding takes time specified in millisecon
      }

      // Sequentially process clothes by passing callback functi
      washClothes(2000, () => {
        dryClothes(3000, () => {
          foldClothes(2000);
        });
      });
    </script>
  </body>
</html>
```

Shortening code

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-s
    <title>Laundry Process</title>
  </head>
  <body>
    <script>
      function processLaundry(taskName, time, callback) {
        setTimeout(() => {
          console.log(`${taskName} complete!`);
          if (callback) callback();
        }, time);
      }

      // Sequentially process laundry tasks
      processLaundry("Washing clothes", 2000, () => {
        processLaundry("Drying clothes", 3000, () => {
          processLaundry("Folding clothes", 2000);
        });
      });
    </script>
  </body>
</html>
```

Callback hell:

The callback hell in JavaScript is referred to as a situation where an excessive amount of nested callback functions are being executed. It reduces code readability and maintenance. The callback hell situation typically occurs when dealing with asynchronous request operations, such as making multiple API requests or handling events with complex dependencies.

Promises

PPT : Link

Layman terms:

- Lets take a Dinner scenario
- Consider a scenario where you and your roommate want to have dinner at home.
- You want to prepare Biryani
- At the same time you feel like having a soft drink from the food truck nearby
- You ask your roommate, “hey can you go down to the food-truck and get soft drink”.
- When he is about to leave, you tell him
    - There is no point waiting till you are back to prepare biryani, so I’ll start with the biryani now but when you reach the place, can you promise that you’ll text me so that I can start setting up the dining table.
    - “Also let me know if something goes wrong. If you can’t find the Food truck or if they are out of soft drinks. Let me know that you can’t get the soft drink and I’ll start cooking some Deserts instead.
- Your friend says “Sure, I promise.I’ll head out now and text you in some time”.
- Now, you go about preparing your biryani but what is the status on softDrink? We can say that it is currently pending till you receive that message from your friend
- When you get back a text message saying that he is getting the soft drink, your desire to have the soft drink has been fulfilled. You can then proceed to set up the table
- If the text message says that he can't bring back any tacos, your desire to have soft drink have been rejected and you now have to cook some deserts instead

Dinner scenario	Javascript
Your friend	Promise
While your friend is on his way to the food truck, you know that it could take a while and you don't want to sit idle. So you start preparing biryani in the meantime	Asynchronous operation in javascript
When your friend text's you with "Can get soft drink/Can't get soft drinks", it answers your question on whether he is getting the it or not	Promise return value
Can get Soft drink	Promise is said to be fulfilled
Cannot get soft drink	Promise is said to be rejected
Set up the table	Success callback
Cook dessert	Failure callback

What is promise?

- A promise is simply an object in JS
- A promise is always one of the three states
  - Pending : which is initial state, neither fulfilled nor rejected.
  - full-filled: meaning that the operation completed successfully
  - rejected: meaning that the operation failed

Why Promises:

- Promises help us deal with asynchronous code in JS

How to create a Promise, How to full-fill or reject, How to execute callback functions based on Promise value

```
const promise = new Promise()
```

- How to fulfil or reject?
    - We need to pass a function to the `Promise Constructor`. That function is called the `executor function` (Remember, fetching the water?). The executor function takes two arguments, `resolve` and `reject`. These two are callback functions for the executor to announce an outcome.

```
let promise = new Promise(function(resolve, reject) {
    // Do something and either resolve or reject
});

console.log(promise) // undefined
```

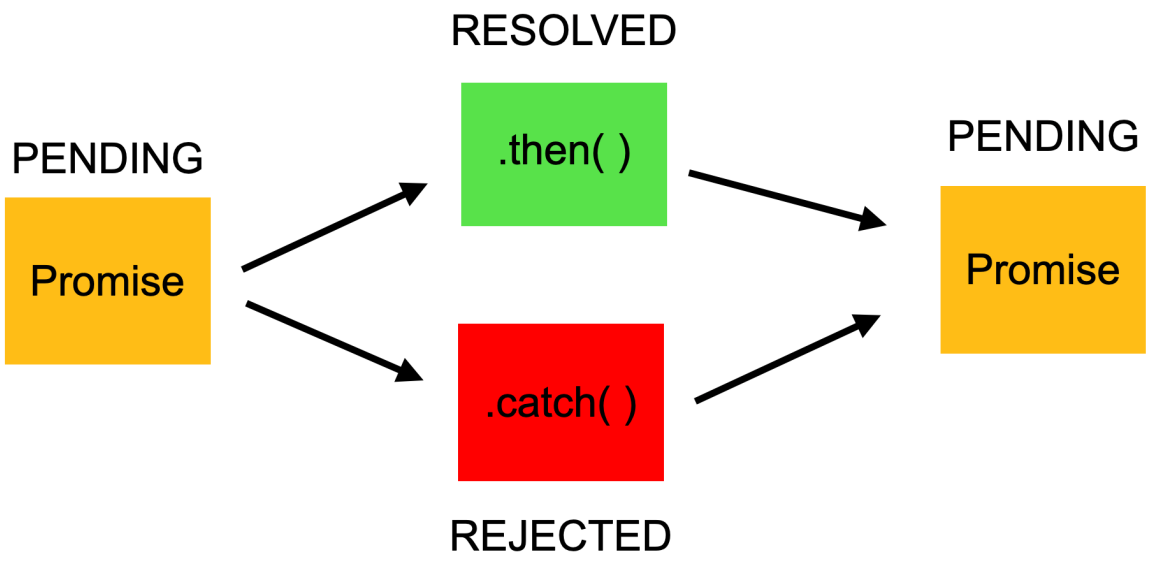
- The `resolve` method indicates successful completion of the task(got drinks), and the `reject` method indicates an error(no drink). You do not implement the resolve/reject method. JavaScript provides it to you. You need to call them from the executor function.
- resolve and reject both are cb fun
- Resolve will be called when change of status from pending to full-filled.
- reject will be called when change of status from pending to rejected.

```
const promise = new Promise((resolve,reject)=>{
    //change of status from pending to fullfilled.
    resolve("Got drink, prepare table")
})

console.log(promise) //value: When the promise is resolved(value
```

```
const promise = new Promise((resolve,reject)=>{
    //change of status from pending to rejected.
    reject("No drink, prepare deserts")
```

```
})

console.log(promise) // error: When the promise is rejected.
```

How to execute callback functions based on status change




```
const promise = new Promise((resolve,reject)=>{
    //change of status from pending to fullfilled.
    resolve("Got drink, prepare table")
})

const promise = new Promise((resolve,reject)=>{
```

```
    //change of status from pending to rejected.
    reject("No drink, prepare deserts")
})


promise.then()
promise.catch()
```

- Promise status : pending to fullfilled ? .then() is executed
- Promise status : pending to rejected ? .catch() is executed
- We get a `.then()` method from every promise. The sole purpose of this method is to let the consumer know about the outcome of a promise. It accepts two functions as arguments, `result` and `error`.

```
const promise = new Promise((resolve,reject)=>{
    //change of status from pending to fullfilled.
    resolve("Brining Soft drinks")
})

const promise = new Promise((resolve,reject)=>{
    //change of status from pending to rejected.
    reject("Not brining Tacos")
})


promise.then((fromResolved)=>{
    console.log(fromResolved)
})
promise.catch((fromReject)=>{
    console.log(fromReject)
})
```

Understanding .then()

Example-1

```
let completedAssignment = new Promise((resolve, reject) => {
    let isCompleted = true;
    if (isCompleted) {
      resolve("Completed assignment");
    } else {
      reject("Not completed assignment");
    }
  });
  console.log(completedAssignment);

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


let completedAssignment = new Promise((resolve, reject) => {
    let isCompleted = false;
    if (isCompleted) {
      resolve("Completed assignment");
    } else {
      reject("Not completed assignment");
    }
  });
  console.log(completedAssignment);

  completedAssignment
    .then((fromResolve) => console.log(fromResolve))
    .catch((fromReject) => console.log(fromReject));
```

Laundry example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-s
    <title>Laundry Process</title>
</head>
<body>
    <script>
        const processTask = (taskName, duration) => new Promise(
            setTimeout(() => {
                console.log(`${taskName} complete!`);
                resolve();
            }, duration);
        });

        processTask('Washing clothes', 2000)
            .then(() => processTask('Drying clothes', 3000))
            .then(() => processTask('Folding clothes', 2000))
            .then(() => console.log("All laundry tasks complete
    </script>
</body>
</html>
```

WE

```
const countValue = 2;

let promiseCount = new Promise(function (resolve, reject) {
    if (countValue>0) {
        resolve("The count value is positive.");
    } else {
        reject("Negative count value!");
    }
```

```
});

console.log(promiseCount);
```

```
const countValue = 2;

let promiseCount = new Promise(function (resolve, reject) {
    if (countValue>0) {
        resolve("The count value is positive.");
    } else {
        reject("Negative count value!");
    }
});


promiseCount
.then(response => console.log(response))
.catch(reason => console.log(reason))
```

```
function wait(t) {
    return new Promise((res, rej) => {
      setTimeout(() => res("Promise Resolved!"), t);
    });
  }
  wait(5000).then((val) => console.log(val));
```

```
// This function returns a new promise that resolves/rejects acc
function checkResult(marks) {
  return new Promise((resolve, reject) => {
    if (marks > 32) {
      resolve("Congrats! You've passed!");
    } else {
      reject(new Error('Failed!'));
    }
```

```
  });
}

checkResult(56)
  .then((result) => {
    console.log(result);
  })
  .catch((err) => {
    console.error(err);
  })
  .finally(() => {
    console.log('Result Received!');
  });

// Output:
// Congrats! You've passed!
// Result Received!
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-s
    <title>Document</title>
  </head>
  <body>
    <input type="number" />
    <button onClick="checkOTP()">Submit</button>
  </body>
</html>

<script>
  function checkOTP() {
    let otp = document.querySelector("input").value;
```

```
    let otpPromise = new Promise((resolve, reject) => {
      setTimeout(() => {
        if (otp == 1234) {
          resolve("correct OTP pin");
        } else {
          reject("incorrect OTP pin");
        }
      }, 2000);
    });

    otpPromise.then((val) => console.log(val)).catch((err) => c
  }
</script>
```