```
// Tasks:

// 1. Write a function to download data from a URL
// 2. Write a function to save that downloaded data in a file & return the fileName
// 3. Write a function to upload the file written in previous step in a newURL.
```

```
// Tasks:

// 1. Write a function to download data from a URL
// 2. Write a function to save that downloaded data in a file & return the fileName
// 3. Write a function to upload the file written in previous step in a newURL.


function writeFile(data, fn) {
    // this function writes data in a new file
    console.log("Started writing data", data);
    setTimeout(function process() {
        console.log("Writing Completed");
        let fileName = "output.txt";
        fn(fileName);
    }, 4000)
}
```

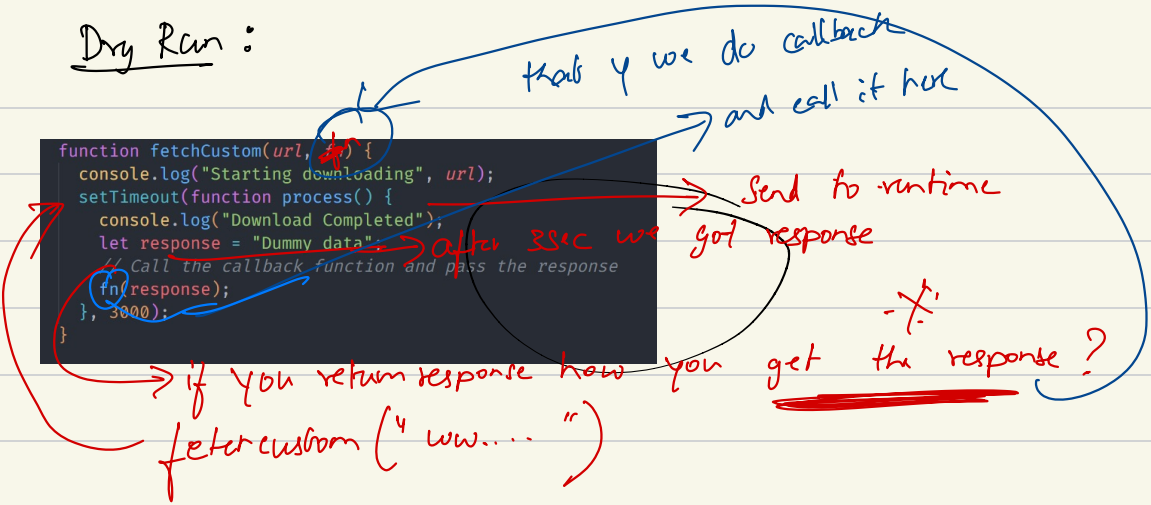
```
function uploadFile(fileName, newURL, fn) {
    console.log("Upload Started");
    setTimeout(function process() {
        console.log("File" + fileName, +  "uploaded successfully on", newURL);
        let uploadResponse = "SUCCESS";
        fn(uploadResponse);
    }, 2000)
}

// passing the response to the user
fetchCustom("www.google.com", function downloadCallBack(response) {
  console.log("Downloaded response is ", response);
  writeFile(response, function writeCallBack(fileNameResponse) {
    console.log("new file written is", fileNameResponse);
    uploadFile(fileNameResponse, "www.drive.google.com", function uploadCallBack(uploadResponse) {
        console.log("Successfully uploaded", uploadResponse);
    })
  })
});
```

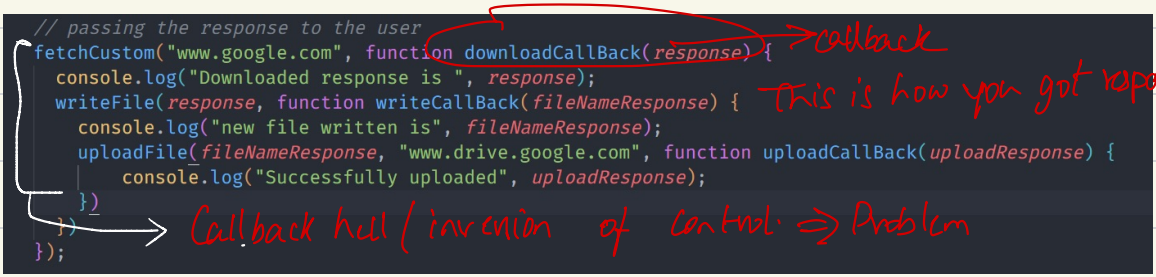
Output:

```
Starting downloading www.google.com
Download Completed
Downloaded response is  Dummy data
Started writing data Dummy data
Writing Completed
new file written is output.txt
Upload Started
Fileoutput.txt NaN www.drive.google.com
Successfully uploaded SUCCESS
```

Using callback

Dry Run :





In similar way for writeFile / uploading the file

```
function writeFile(data, fn) {
    // this function writes data in a new file
    console.log("Started writing data", data);
    setTimeout(function process() {
        console.log("Writing Completed");
        let fileName = "output.txt";
        fn(fileName);
    }, 4000)
}
```

Using promise

Let's make our own promise :

promiseDemo2.js

03 - Async JS > Promises > promiseDemo2.js > fetch

```
1  // how can we write a function to download some data from URL, and not use callbacks
2  // instead use promises ?
3
4  function fetch(url){
5      return new Promise(function () {
6
7      });
8  }
```

new promise object ⇒ constructor
- If take one parameter which is function

- If return new promise (), it's a promise constructor
- If take one parameter which is a function
- The function has two parameter which is resolve and reject.

Syntax:

Promise constructor ⇒ if take call back as argument

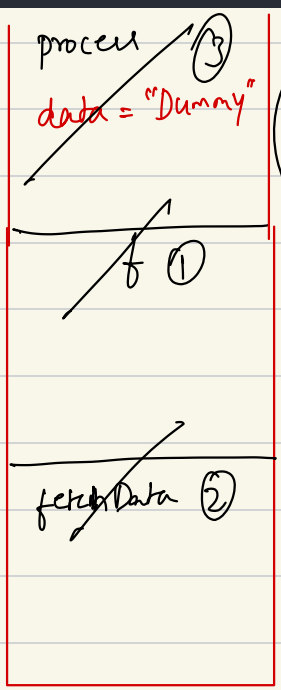
```
return new Promise(function(resolve, reject){

});
```

Code :

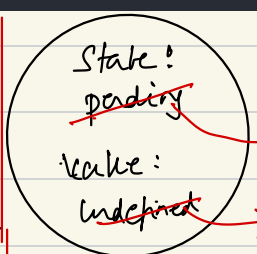
```
function fetchData(url){
    return new Promise(function((resolve, reject) {
        console.log("going to start download");
        setTimeout(() => {
            let data = "Dummy Downloaded Data";
            console.log("Download Completed")
            resolve(data);
        }, 10000);
        console.log("Timer to minic download started")
    });
}

console.log("Starting the program")
console.log("We are trying to minic a downloader")
x = fetchData("www.google.com");
console.log("New promise object created successfully, but downloading still going on")
```

Runtime

function process()



Call Stack



global

x

O/P

- Starting the program
- We are trying to minic a down loader
- going to start download
- Timer to minic download started
- New promise object.....
- Download completed

- Now, the question is, will JS wait ? (Line no # 39)

Store the promise obj

function returns a promise object

Will JS wait here for promise to be resolved if it involves any async piece of code ?

- setTimeout → async (Yes async, so it will go to runtime, so it will NOT WAIT)
- for loop → synchronous (No async code, so technically it will wait) → has to wait

So while creating promise object in for loop it's still getting created, now once compled the for loop then the promise object is created and it's by default fulfilled.

```
> function fetchData(url) {
      return new Promise(function (resolve, reject) {
          console.log("Started downloading from the url", url);
          // setTimeout(function processDownloading() {
          //     let data = "Dummy Data";
          //     console.log("Download Completed");
          //     resolve(data);
          // }, 7000);
          for(let i = 0; i < 10000000; i++){}
          resolve("dummy data");
      });
  };
< undefined
```

this callback is having a long sync piece of code so JS will have to wait for promise object creation. and just after the for loop we resolve the promise, so we get resolved promises.

```
> x = fetchData();
  console.log("done");
  Started downloading from the url undefined
  done
< undefined
> x
< ▼ Promise {<fulfilled>: 'dummy data'} i
    ▶ [[Prototype]]: Promise
      [[PromiseState]]: "fulfilled"
      [[PromiseResult]]: "dummy data"
>
```

To BRIEF OUT:

— If creation of promise involves sync piece of code it will wait OR NOT

```
function fetchData(url) {
    return new Promise(function (resolve, reject) {
        console.log("Started downloading from the url", url);
        setTimeout(function processDownloading() {
            let data = "Dummy Data";
            console.log("Download Completed");
            resolve(data);
        }, 7000);
        resolve("dummy data");
    });
};
```

→ Promise object will get created easily as there is NO blocking piece of code

but initally it will be pending as the fulfillment happen after a timer of 7 sec.

2:18

- Now technically, when promise gets resolved, we have to execute some function

→ We can use `.then()` function on the promise object to bind the function we want to execute once we fulfill a promise

- The `.then()` function takes function as an argument that we want to execute after promise fulfills, and the argument function takes value property as a parameter.

```
// Tasks:
// 1. Write a function to download data from a URL.
// 2. Write a function to save that downloaded data in a file & return the fileName
// 3. Write a function to upload the file written in previous step in a newURL.

function fetchData(url) {
    return new Promise(function (resolve, reject) {
        console.log("Started downloading from the url", url);
        setTimeout(function processDownloading() {
            let data = "Dummy Data";
            console.log("Download Completed");
            resolve(data);
        }, 7000);
        [redacted]
    });
};

function writeFile(data) {
    return new Promise(function (resolve, reject) {
        console.log("Started writing ", data, " in a file");
        setTimeout(function processWriting() {
            let fileName = "result.txt";
            console.log("File written successfully")
            resolve(fileName);
        }, 3000)
    });
};
```

```
function uploadData(fileName, url) {
    return new Promise(function (resolve, reject) {
        console.log("Upload started on url", url, " file name is", fileName);
        setTimeout(function processUpload() {
            let result = "SUCCESS";
            console.log("Uploading done");
            resolve(result);
        }, 5000)
    });
};

// let data = fetchData("www.datadrive.com");
// let fileName = writeFile(data);
// let response = uploadData(fileName, "www.googledrive.com");
// the above code will not work in required fashion
```

Now the function goes step by step

```
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise.then(function processDownload(value) {
    console.log("Download promise fulfill");
    let writePromise = writeFile(value);
    writePromise.then(function processWrite(value) {
        console.log("Wrting a file completed");
        console.log(value);
    })
})
```

this will be stored in an array on Fulfillment.

```
Started downloading from url www.datadrive.com
Download Completed
Download promise fulfill
Started writing  Dummy Data  in a file
File written successfully
Wrting a file completed
result.txt
```

Benefit of using Promises instead of callbacks:

— Now the processDownload ive are not passing to the function "fetchData" were some one else has implemented. (You are never giving control of processDownload inside fetchData) So problem of inversion control is solved.

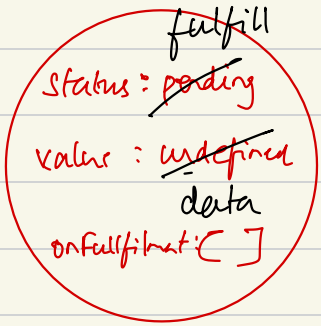
after line #13, if you write multiple

resolve("abc")
resolve(42) // this thing doesn't executed

- those lines will not be executed as promise is fulfilled once.

```
function fetchData(url) {
  return new Promise(function (resolve, reject) {
    console.log("Started downloading from url", url);
    setTimeout(function processDownloading() {
      let data = "Dummy Data";
      console.log("Download Completed");
      resolve(data);
      console.log("hello");

    }, 7000);
  });
}
```

once it hit this line it change the status & update the value




```
Started downloading from url www.datadrive.com
Download Completed
hello
Download promise fulfill
```

Still in the callback you have #13 to execute so, it will print "hello"

```
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise.then(function processDownload(value) {
  console.log("Download promise fulfill");
//   let writePromise = writeFile(value);
//   writePromise.then(function processWrite(value) {
//     console.log("Wrting a file completed");
//     console.log(value);
//   });
});
```

this callback are stored in onfulfillment array.

- then() → only work for resolve, not reject

```
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise.then(function processDownload(value) {
  console.log("Download promise fulfill");
  let writePromise = writeFile(value);
  writePromise.then(function processWriting(value) {
    console.log("Wrting done");
    console.log("file name is ", value);
  });
});
```

Promise hell

```
Started downloading from url www.datadrive.com
Download Completed
Download promise fulfill
Started writing  Dummy Data  in a file
File written successfully
Wrting done
file name is  result.txt
```

- But still callback hell is present, it's also known as promise hell.
- Above code solves inversion of control but not promise hell.

```
55  let downloadPromise = fetchData("www.datadrive.com");
56  downloadPromise
57  .then(function processDownload(value) {
58      console.log("Downloading is done with the following value", value);
59      return "Rizon";
60  })
```

this is a function so we can return

```
Started downloading from url www.datadrive.com
▼ Promise {<pending>} i
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "Rizon"
Download Completed
Downloading is done with the following value Dummy Data
```

Now if you return you see the value got updated to "Rizon" State is fulfilled

Now if you don't return then it's undefined.

```
downloadPromise
.then(function processDownload(value) {
    console.log("Downloading is done with the following value", value);
    // return "Rizon";
})
Started downloading from url www.datadrive.com
▼ Promise {<pending>} i
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: undefined
Download Completed
Downloading is done with the following value Dummy Data
```

Now if you comment .then() it's Dummy Data (expected behaviour)

```
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise
// .then(function processDownload(value) {
//     console.log("Downloading is done with the following value", value);
//   // return "Rizon";
// })
Started downloading from url www.datadrive.com
▼ Promise {<pending>} i
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "Dummy Data"
```

if you attach .then() you see new Promise is created.

```
let downloadPromise = fetchData("www.datadrive.com");
let x = downloadPromise
.then(function processDownload(value) {
    console.log("Downloading is done with the following value", value);
   return "Rizon";
})
```

```
Started downloading from url www.datadrive.com
undefined
Download Completed
Downloading is done with the following value Dummy Data
> downloadPromise
Promise {<fulfilled>: 'Dummy Data'}
  [[Prototype]]: Promise
  [[PromiseState]]: "fulfilled"
  [[PromiseResult]]: "Dummy Data"
> x
Promise {<fulfilled>: 'Rizon'}
  [[Prototype]]: Promise
  [[PromiseState]]: "fulfilled"
  [[PromiseResult]]: "Rizon"
```

- downloadPromise = fetchData("www.google.com")
- downloadPromise.then(function f(value){
    console.log(value)
    return "RIZON";
  })

the .then function itself return a new promise

```
let downloadPromise = fetchData("www.datadrive.com");
x = downloadPromise
.then(function processDownload(value) {
    console.log("Downloading is done with the following value", value);
    return "Rizon";
})

x.then(function processDownload(value) {
    console.log("x promise value is", value);
})
```

```
Started downloading from url www.datadrive.com
Download Completed
Downloading is done with the following value Dummy Data
x promise value is Rizon
```

```
let downloadPromise = fetchData("www.datadrive.com");
x = downloadPromise
.then(function processDownload(value) {
    console.log("Downloading is done with the following value", value);
    // return "Rizon";
})

x.then(function processDownload(value) {
    console.log("x promise value is", value);
})
```

if no return then undefined that what you saw above.

```
Started downloading from url www.datadrive.com
Download Completed
Downloading is done with the following value Dummy Data
x promise value is undefined
```

For the cleaner code:

```
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise
.then(function processDownload(value) {
    console.log("Downloading done with the following value", value);
    return value;
})
```

Let's complete the whole code:

```
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise
    .then(function processDownload(value) {
        console.log("Downloading done with the following value", value);
        return value;
    })
    .then(function processWrite(value) {
        return writeFile(value);
    })
    .then(function processUpload(value) {
        return uploadData(value, "www.googledrive.com");
    })

```

return new Promise

consumed here and return new Promise

consumed here and return new promise

• then() ⇒ consume promise and return new promise

```
Started downloading from url www.datadrive.com
Promise {<pending>}
    [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: "SUCCESS"
Download Completed
Downloading done with the following value Dummy Data
Started writing  Dummy Data  in a file
File written successfully
Upload started on url www.googledrive.com  file name is result.txt
Uploading done
```

event loop

Priority Queue

micro task queue (Read about it)

• then()

promised call back are in micro task queue.