Async and Await

What is an Asynchronous operation?

When you are going to a heavy operation. File read/ API call
This would be done asynchronously.

Async : When you have async written in front of a function the function will always return a promise wrapping whatever was getting returned from the function.

```
async function fetchData() {
	return ‘data’;
}

const dataPromise = fetchData();

async function abc() {

}

var a = abc();
```

Returns undefined wrapped with a promise.

```
const p = new Promise(function(resolve , reject){
  resolve('Promise Resolved')
})


async function fetchData() {
  return p;
}

const dataPromise = fetchData()
console.log(dataPromise);

dataPromise.then((res)=> {console.log(res)});
```

If already a promise is getting returned it wont change anything but otherwise it will append a promise

Await

```
const p = new Promise((resolve , reject)=>{
  resolve('Promise Resolved')
})
```

```
function fetchData() {
  p.then((res)=> console.log(res))
}

fetchData()
```

Async/Await

```
const p = new Promise((resolve , reject)=>{
  resolve('Promise Resolved')
})

async function handlePromise() {
        const val = await p;
        console.log(val);
}

handlePromise();


//async function handlePromise() {
  const val = await p;
  console.log(val)
}

p.then((res) => {
        console.log(val);
        console.log(2);
})
```

Adding Async behaviour

```
const p = new Promise((resolve , reject)=>{
  setTimeout(()=>{
      resolve('Promise Resolved')
  } , 10000)
})

function fetchData() {
  p.then((res)=> console.log(res))
  console.log("Create Impact")
}

 fetchData()
```

```
const p = new Promise((resolve , reject)=>{
  setTimeout(()=>{
      resolve('Promise Resolved')
  } , 10000)
})

async function handlePromise() {
  const val = await p;
  console.log('Create Impact'
  console.log(val);
}



 handlePromise();
```

Next question

```
const p = new Promise((resolve , reject)=>{
  setTimeout(()=>{
      resolve('Promise Resolved')
  } , 10000)
})
```

```
async function handlePromise() {
  const val = await p;
  console.log('Create Impact');
  console.log(val);

  const val2 = await p;
  console.log('Create Impact 2');
  console.log(val2);
}

handlePromise();
```

Next question

```
const p1 = new Promise((resolve , reject)=>{
  setTimeout(()=>{
      resolve('Promise Resolved')
  } , 10000)
})

const p2 = new Promise((resolve , reject)=>{
  setTimeout(()=>{
      resolve('Promise Resolved')
  } , 5000)
})



async function handlePromise() {
  // JS engine waits for the promise to get resolved and then
moves forward

  console.log("Scaler")

  const val = await p1

  console.log('Create Impact 1')
  console.log(val)

  const val2 = await p2

  console.log('Create Impact 2')
  console.log(val2)
}

handlePromise()
```

Coffee Shop example

```
function placeOrder(drink) {
    return new Promise(function(resolve, reject) {
        if(drink === 'coffee') {
            resolve('Order for Coffee Placed.')
        }
        else {
            reject('Order can not be Placed.')
        }
    })
}


placeOrder(‘coffee’).then((orderStatus)=> {
        console.log(orderStatus);
}).catch(function(error) {
    console.log(error)
})



function processOrder(orderPlaced) {
        return new Promise(function(resolve) {
                resolve(‘${orderPlaced} and Served’);
        })
}
```

```
placeOrder('coffee').then(function(orderStatus) {
    console.log(orderStatus)
    return orderStatus
}).then(function(orderStatus) {
    let orderIsProcessed = processOrder(orderStatus)
    console.log(orderIsProcessed)
    return orderIsProcessed
}).then(function(orderIsProcessed) {
    console.log(orderIsProcessed)
})


function generateBill(processedOrder) {
    return new Promise(function(resolve) {
        resolve(`${processedOrder} and Bill Generated with 200
Rs.`)
    })
}


placeOrder('coffee').then(function(orderStatus) {
    console.log(orderStatus)
    return orderStatus
}).then(function(orderStatus) {
    let orderIsProcessed = processOrder(orderStatus)
    console.log(orderIsProcessed)
    return orderIsProcessed
}).then(function(orderIsProcessed) {
    console.log(orderIsProcessed)
    return orderIsProcessed
}).then(function(orderIsProcessed) {
    let BillGenerated = generateBill(orderIsProcessed)
    return BillGenerated
}).then(function(BillGenerated) {
    console.log(BillGenerated)
}).catch(function(err) {
    console.log(err)
})
```

With async await

```
async function serveOrder(){
        let orderstatus = await placeOrder('coffee')
        console.log(orderstatus)
        let processedOrder = await processOrder(orderstatus)
        console.log(processedOrder)
        let generatedBill = await genreateBill(processedOrder)
        console.log(generatedBill)
}
```

Catch

```
async function serveOrder(){
    try {
        let orderstatus = await placeOrder('tea')
        console.log(orderstatus)
        let processedOrder = await processOrder(orderstatus)
        console.log(processedOrder)
        let generatedBill = await genreateBill(processedOrder)
        console.log(generatedBill)
    } catch (error) {
        console.log(error)
    }
}
```