NODE

ASYNCHRONOUS JAVASCRIPT

Problem

```
console.log("Before...");
setTimeout(function () {
 console.log("Reading a user from DB");
 }, 2000);
console.log("After...");
// Whats the output
```

Output of Problem

Before...

After...

Reading a user from DB

ASynchronous

Its not

- Concurrent
- Multi Threaded

It is

- Just a function scheduled to be called in future

Output

```
console.log("Before");
const user = getUser();
console.log(user);
console.log("After");
//Output
//Before
//dummy
//After
//DB Query entertained
```

```
function getUser(){
 setTimeout(function(){
 console.log('DB Query entertained');
 return {id:9,name:'usman'}
 },1000);
return "dummy";
}
```

Patterns for Dealing with Asynchronous Code

Callback

Promises

Async/await

CallBack

```
function getUser(id, callback) {
 setTimeout(function () {
  console.log("Reading User");
  callback({ id: id, name: "Usman" });
 }, 2000)
}
```

CallBack

```
console.log("Before");
getUser(1, function (userObj) {
 console.log("Received User");
 console.log(userObj);
});
console.log("After");
```

Imagine this ☹ (CallBack Hell)

```
getUser(1, (user) => {
  getRepositories(user.gitHubUsername, (repos)      =>
{
    getCommits(repos[0], (commits) => {
    console.log(commits);
    })
  })
});
//You can use named Functions but still NOOOOO
```

Promise

```
const p = new Promise(function(resolve,reject){
resolve({name:"hareem"});
reject(new Error("Hareem is naughty"));
});
p.then((result)=>{
console.log(result.name);
});
p.catch((error)=>{console.log("Error Caught"+error.message)});
```

Promise

```
const p = new Promise((resolve, reject) => {
 // Kick off some async work
 // ...
 setTimeout(() => {
 resolve(1); // pending => resolved, fulfilled
 reject(new Error('message')); //pending => rejected
 }, 2000);
});
```

Using Promise

```
 p.then(result => console.log('Result',
result))
 .catch(err => console.log('Error',
err.message));
```

Beauty in Code

```
getUser(1)
.then(user => getRepositories(user.gitHubUsername))
.then(repos => getCommits(repos[0]))
.then(commits => console.log('Commits', commits))
.catch(err => console.log('Error', err.message));
```

Async and Await approach

```
async function displayCommits() {
 try {
  const user = await getUser(1);
  const repos = await getRepositories(user.gitHubUsername);
  const commits = await getCommits(repos[0]);
  console.log(commits);
 }
 catch (err) {
  console.log('Error', err.message);
 }
}
```

Resolved Promises (For Testing)

```
Promise.reolve(1);
Promise.reject(new Error(''));
```

Running Promses in parallel

```
Promise.all([p1, p2]);// When all promises are resolved

Promise.race([p1, p2]);// When any one finished first
```

Sample Code

https://1drv.ms/f/s!AtGKdbMmNBGd0V1N14IfBGU1Npoi