ASYNC AWAIT

Before version 7.6, callbacks were the only way to run function one after the other.

```
app.get('/', function(){
    function1(arg1, function(){
        ...
    })
});
```

The problem with this kind of code is that this kind of situation can cause a lot of trouble and the code can get messy when there are several functions. The situation is known as Callback Hell.

In order to resolve Callback Hell, promises and function chaining was introduced.

```
function fun(req, res) {
    return request.get('http://localhost:3000')
    .catch((err) => {
        console.log('found error');
    }).then((res) => {
        console.log('get request returned.');
});
```

The example above if a demonstration of function chaining instead of callbacks. It can be observed that the code is more readable and easy to understand.

If it needs to be made sure that the functions are running in the desired sequence chain them one after another.

In Node version 8, async/await feature was introduced. In this the function need not be chained after one another but simply await the function.

- The await function returns a promise.
- Async function needs to be declared before awaiting a function returning a promise.

```
async function fun(req, res){
    let response = await request.get('http://localhost:3000');
    if (response.err) { console.log('error');}
    else { console.log('fetched response');
}
```

- The code above basically wait for the request.get() function to complete before moving on to the next line to execute it.
- The request.get()function returns a Promise for which user will await .