

# JavaScript Asynchronous Patterns

---

Async Await

**then**

# Agenda

- Async Await
- Further/advanced async behaviour



# Recap -Javascript Characteristics

- JavaScript is single threaded
- JavaScript is event driven
  - Events happen – we write code to deal with them
  - Can use callbacks to do this
- JavaScript can be Asynchronous
  - Order of operation results may differ from order they were called...

A large, dark gray 'JS' logo is centered on a solid yellow rectangular background. The letters are bold and sans-serif.

Callbacks (again)

# What have Callbacks ever done for us...

- Great for things that can happen multiple times
- Great if you don't really care about what happened before you attached the listener
- Great if it's a straight-forward, stand-alone event with a quick resolution time
- Great if callback is not part of sequential process.

E.g. key press event on control.

```
document.getElementById("demo").addEventListener("keypress", myFunction);
```

- **But...**



# Multiple Callbacks

I want this to happen sequentially... but order of callback events is indeterminate.

```
console.log("about to read...");
fs.readFile("test-1.txt", "utf8", (err, contents) => {
  console.log(contents);
});
fs.readFile("test-2.txt", "utf8", (err, contents) => {
  console.log(contents);
});
fs.readFile("test-3.txt", "utf8", (err, contents) => {
  console.log(contents);
});
console.log("...done");
```

Possible Result

```
about to read...
...done
- 4
- 5
- 6
- 7
- 8
- 9
- 1
- 2
- 3
```

Possible Result

```
about to read...
...done
- 1
- 2
- 3
- 7
- 8
- 9
- 4
- 5
- 6
```

# Callback Hell – Multiple sequential requests

```
callback-hell.js
1 var amount=req.param("amount");
2 db.select("* from sessions where session_id=?", req.param("session_id"), function(err, sessions) {
3   if (err) throw err;
4   db.select("* from accounts where user_id=?", sessions[0].user_ID, function(err, accounts) {
5     if (err) throw err;
6     if (accounts[0].balance < amount) throw new Error('insufficient balance');
7     db.execute("withdrawal(?, ?)", accounts[0].ID, req.param("amount"));
8     if (err) throw err;
9     res.write("withdrawal OK, amount: " + req.param("amount"));
10    db.select("balance from accounts where account_id=?", accounts[0].ID, function(err, balance) {
11      if (err) throw err;
12      res.end("your current balance is " + balance.amount);
13    });
14  });
15 });
16 }
```

```
1
2
3 function loadUpThatApplication() {
4   request("/api/getCustomer", function(response){
5     var customerId = response.customer.id;
6     request("/api/customer/accounts/"+customerId, function (response2) {
7       request("http://facebook/pics/"+response2.faceBookUserName, function (response3) {
8         showTheUserThatBeautifulUI(response3, function () {
9           byeByeSpinner();
10         });
11       });
12     });
13   });
14 }
15
16
```

# Async Functions

using `async/await`



# Async/Await !

- **async/await** and **promises** are essentially the same under the hood
- **async** is a keyword
  - Used in function declaration
- **await** is used during the promise handling
  - must be used within an **async function**
- **async** functions return a promise, regardless of what the return value is within the function.
- **Available now!** in most good browsers as well as Node.js

```
import {promises as fs} from 'fs';

console.log("about to read...");

readFiles()

console.log("...doing other stuff")

|

async function readFiles() {
  console.log("starting sequential read...");
  const contents1 = await fs.readFile("test-1.txt", "utf8");
  console.log(contents1);
}
```

# Async/Await Sequential read

```
console.log("about to read...");

readFiles()

console.log("...doing other stuff")

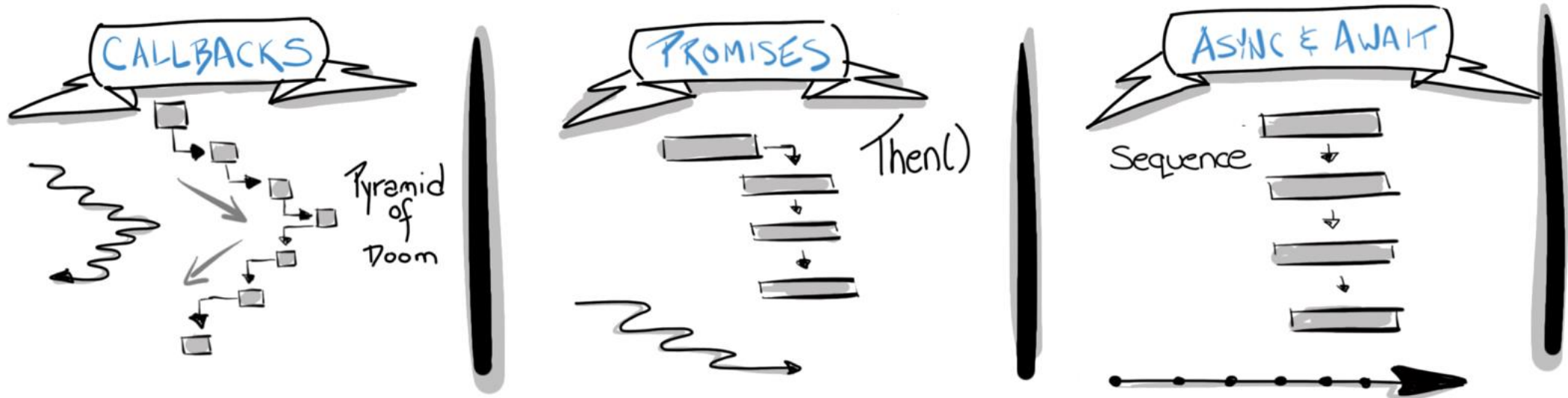
async function readFiles() {
  console.log("starting sequential read...");
  const contents1 = await fs.readFile("test-1.txt", "utf8");
  console.log(contents1);
  const contents2 = await fs.readFile("test-2.txt", "utf8");
  console.log(contents2);
  const contents3 = await fs.readFile("test-3.txt", "utf8");
  console.log(contents3);
  console.log("...done sequential read");
}
```

output

```
about to read...
starting sequential read...
...doing other stuff
(node:1988) ExperimentalWarning:
- one
- two
- three
- four
- five
- six
- seven
- eight
- nine
...done sequential read
```

# Callbacks vs Promises vs Async-Await

Asynchronous Javascript



# Error Handling

# Error Handling – async await

Use try -catch

```
async function readFiles() {  
  try{  
    console.log("starting sequential read...");  
    const contents1 = await fs.readFile("test-1.txt", "utf8");  
    console.log(contents1);  
    const contents2 = await fs.readFile("test-22.txt", "utf8");  
    console.log(contents2);  
    const contents3 = await fs.readFile("test-3.txt", "utf8");  
    console.log(contents3);  
  }catch (error){  
    console.error("failed to read a file!", error)  
  }  
  console.log("...done sequential read");  
}
```

Further Asynchronous features...

# Wrapper Function for Error handling

- Express can't handle promise errors/rejections
- An Async function always returns a Promise.
  - can *wrap* the async function to catch errors in Express Apps
  - Can drop try/catch in every async function
- Makes code more readable?

```
import asyncHandler from 'express-async-handler';

const router = express.Router();

// Get all users
router.get('/', asyncHandler(async (req, res) => {
  const users = await User.find();
  res.status(200).json(users);
})));
```

# Parallelism //

- Some processes need to be sequential
  - Eg. Had to get data back from API **BEFORE** getting link URL
- REMEMBER: Should only be sequential if you need to be...

Takes 1000ms

```
async function series() {  
  await wait(500); // Wait 500ms...  
  await wait(500); // ...then wait another 500ms.  
  return "done!";  
}
```

Takes ~500ms

```
async function parallel() {  
  const wait1 = wait(500); // Start a 500ms timer asynchronously...  
  const wait2 = wait(500); // this timer happens in parallel.  
  await wait1; // Wait 500ms for the first timer...  
  await wait2; // ...by which time this timer has already finished.  
  return "done!";  
}
```



# Sources

- <https://developers.google.com/web/fundamentals/primers/promises>
- <https://stackoverflow.com/questions/2069763/difference-between-event-handlers-and-callbacks>
- <https://medium.com/@Abazhenov/using-async-await-in-express-with-node-8-b8af872c0016>