Practical Promises

Let’s keep it practical

Asynchronous Code

What is asynchronous code?

- Asynchronous aka “async” operations in JavaScript take time or happen in the future, rather than immediately.
- JavaScript doesn't wait for async operations to finish before moving on and continues executing other tasks concurrently.
- Asynchronous programming allows for efficient and non-blocking execution.
- It's useful for handling time-consuming tasks and making programs more responsive.
- JavaScript provides callbacks, promises, and async/await to manage async operations.
- Async programming enables tasks like fetching data or performing calculations without freezing the program.

Async Code Practically

In which order will the logs fire?

```
console.log("One")
setTimeout(() => console.log("Two"), 10)
console.log("Three")
```

Async Code Practically

```
console.log("One")
setTimeout(() => console.log("Two"), 10)
console.log("Three")
```

Obviously One will be first, but what will be next?

1) One

Async Code Practically

```
console.log("One")
setTimeout(() => console.log("Two"), 10)
console.log("Three")
```

Next up is Three! Surprise Surprise

1) One
2) Three

Async Code Practically

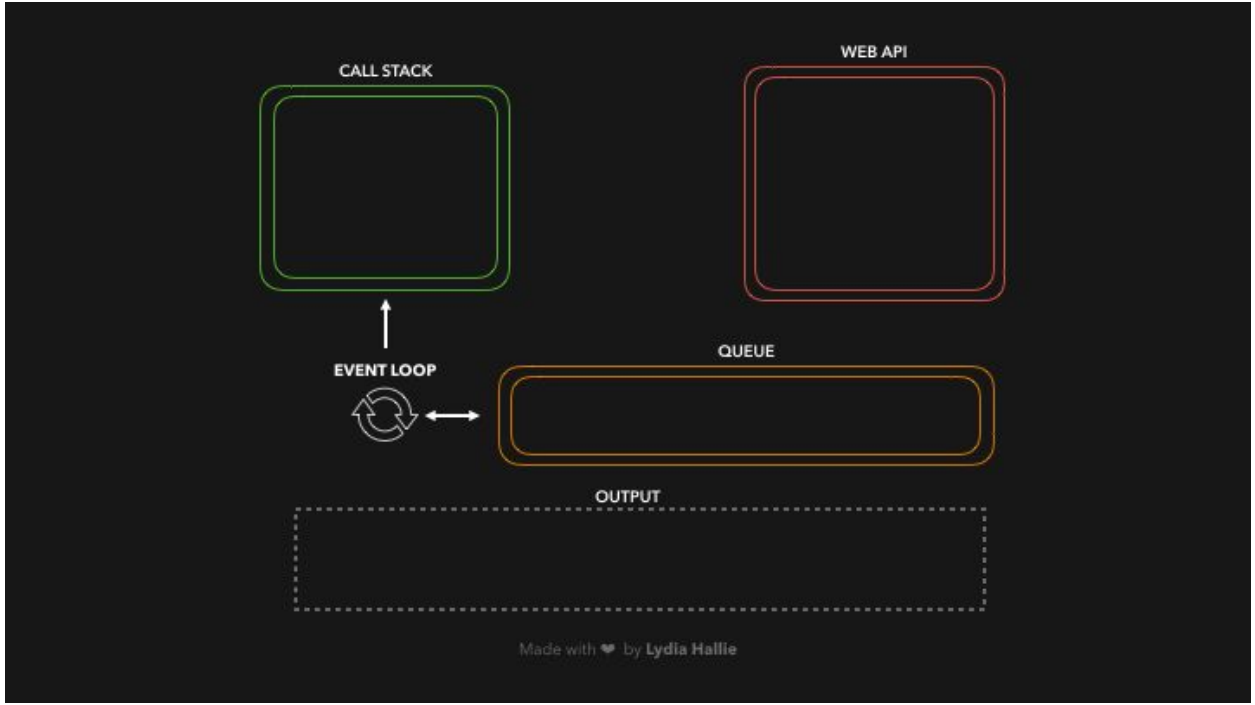
```
console.log("One")
setTimeout(() => console.log("Two"), 10)
console.log("Three")
```

So that leaves Two

1) One
2) Three
3) Two

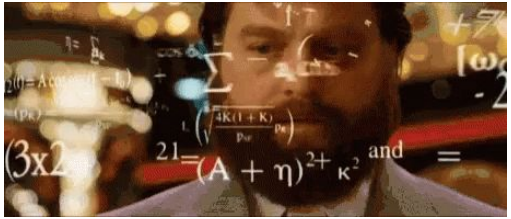
Under the Hood - Event Loop

The event loop in JavaScript is a mechanism that manages the execution of code and handles asynchronous operations, ensuring responsiveness without blocking the main thread.



Handling Async Code

How do we handle Async code?



1. Callbacks … But what is a callback?

What is a callback? - Refresher

- A function passed to another function.

- It comes in two flavors:
  - Blocking
  - Non-blocking

Async with Callbacks

```
console.log("Getting Configuration")
fs.readFile('/config.json', 'utf8', (err, data) => {
console.log("Got configuration:", data)
});
console.log("Moving on...");
```

Which order will the logs fire here?

The problems with callbacks

```
const tryGetRich = () => {
  readFile("/luckyNumbers.txt", (err, fileContent) => {
    // Do something with lucky numbers
  });
};
```

Pretty simple huh? But what if we want to do something with the fileConent?

The problems with callbacks

```
const tryGetRich = () => {
  readFile("/luckyNumbers.txt", (err, fileContent) => {
    nums = fileContent.split(",");
    nums.forEach((num) => {
      bookmaker.getHorse(num, (err, horse) => {
        // Ok, this is getting a little confusing
      });
    });
  });
};
```

Ok, so we are getting a little more intense here...

The problems with callbacks

```
const tryGetRich = () => {
  readFile("/luckyNumbers.txt", (err, fileContent) => {
    nums = fileContent.split(",");
    nums.forEach((num) => {
      bookmaker.getHorse(num, (err, horse) => {
        bookmaker.bet(horse, (err, success) => {
          if (success) {
            // Help...
          }
        });
      });
      console.log("When will I run??");
    });
  });
};
```

We enter something called callback hell!

Wrong

```
var result;
setTimeout(function cb() {
  result = "hello";
}, 0);
console.log(result);
```

- Result will most likely be undefined.
- The log result will print before the setTimeOut promise resolves.

Wrong Again

```
var result = setTimeout(function cb() {
  return "hello";
}, 0);
console.log(result);
```

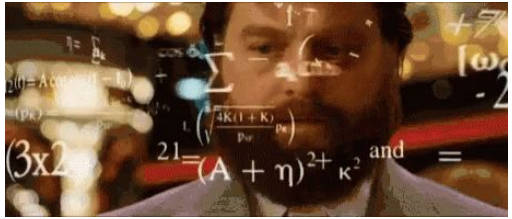
- The output of this will be a number.
- The number is the timer ID returned by setTimeOut

Correct

```
setTimeout(function cb() {
  var result = "hello";
  console.log(result);
}, 0);
```

- By logging the result variable within the callback function, you ensure that 'hello' is outputted when the callback is executed.

How do we handle Async code?



1. Callbacks … Ok we got this down
2. Promises … But now what are these?!

promise

“A promise represents the eventual result of an asynchronous operation.”

– The Promises/A+ Spec

Callback vs Promises

Vanilla Async callback

```
readFile("file.txt", function (err, fileContent) {
  // Do something here
});
```

Async Promises

```
fs.readFileAsync("file.txt").then(
  function onSuccess(data) {
    // Handle the successful reading of the file
    console.log(data); // Example: Output the data to the console
  },
  function onError(err) {
    // Handle the error that occurred during file reading
    console.error(err); // Example: Output the error message to the console
  }
);
```

Stages of Promises

1. **Pending:** This is the initial stage when the promise is created. At this stage, the promise is neither fulfilled nor rejected. It is "pending" and waiting for the asynchronous operation to complete.

2. **Fulfilled(Settled):** When the asynchronous operation associated with the promise is successfully completed, the promise transitions to the "fulfilled" state. At this stage, the promise is considered resolved, and the associated data (the result of the asynchronous operation) is available.

3. **Rejected(Settled):** If an error occurs during the asynchronous operation, the promise transitions to the "rejected" state. This indicates that the operation has failed, and an error object is provided as the reason for the rejection.

Wrong

```
var result;
promisifiedSetTimeout(0).then(function success() {
  result = "hello";
});
console.log(result);
```

- Result will most likely be undefined.
- The log result will print before the setTimeOut promise resolves.

Wrong Again

```
var result = promisifiedSetTimeout(0).then(function success() {
  return "hello";
});
console.log(result);
```

- The output of this will be the promise object.

{

[[PromiseValue]]: undefined,

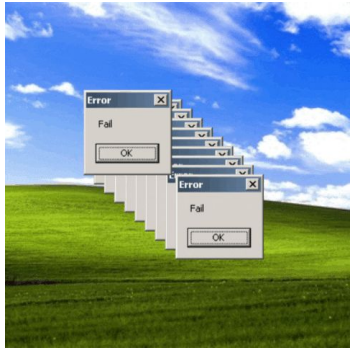
[[PromiseStatus]]: "pending"

}

Correct

```
var result = promisifiedSetTimeout(0)
  .then(function success() {
    return "hello";
  })
  .then(function handleResult(value) {
    console.log(value); // Output: 'hello'
  });
```

- Our first then handles the return of the promises
- Our second then handles logging it.



Error Handling

How do we handle errors?

We can get errors for numerous reasons, but it is important we write out code to handle these situations.

Possible HTTP Error Codes:

- 401 Unauthorized
- 404 Not Found
- 500 Server Error
- Etc ...

Error Handling

Fine

```
var path = "demo-poem.txt";

promisifiedReadFile(path).then(
  function (buff) {
    console.log(buff.toString());
  },
  function (err) {
    console.error(err);
  }
);
```

Better

```
var path = "demo-poem.txt";

promisifiedReadFile(path)
  .then(function (buff) {
    console.log(buff.toString());
  })
  .then(null, function (err) {
    console.error(err);
  });
```

-

Best

```
var path = "demo-poem.txt";

promisifiedReadFile(path)
  .then(function (buff) {
    console.log(buff.toString());
  })
  .catch(function (err) {
    console.error(err);
  });
```

-

Promises Advantages

1. **Readability and Maintainability:** Promises provide structured code that is easier to understand and maintain.
2. **Error Handling:** Promises offer built-in error handling mechanisms for synchronous and asynchronous errors.
3. **Sequential and Parallel Execution:** Promises enable sequential or parallel execution of asynchronous operations.
4. **Promise Composition:** Promises can be composed together for complex asynchronous workflows.
5. **Error Propagation:** Errors in promises automatically propagate through the chain, simplifying error handling.
6. **Promise-based APIs:** Many libraries and frameworks offer promise-based APIs for easier integration and compatibility.