5. Asynchronous Control Flow

Asynchronous Programming

Github * JavaScript is `asynchronous`; executing one line of code after the previous one has completed * Asynchronous programming allows us to start a long running process, execute other code, and then perform some operation once the running long process has completed * We generally use async programming when using API's outside of our application (eg. interacting with a database, making a web request, retrieving the user's geolocation coordinates) * **Note:**Using a callback does not necessarily mean asynchronous code. Think of a `forEach` or a `map`. That code is executed synchronously while using a callback.

Blocking vs Non-Blocking

- Code that takes a long time to finish and stops other code from executing is called `blocking` code
- Code that takes a long time to finish, but doesn't affect the code around it is called `non-blocking`

`setTimeout` and `setInterval`

- The `setTimeout` function allows us to wait a specific number of milliseconds before executing some code

```
// setTimeout takes 2 arguments:
// * a callback
// * an integer representing the number of ms to wait before firing the
callback
setTimeout(() => {
  console.log('hello world'); // prints "hello world" to the console
after 2 seconds
}, 2000);
```

- Code around a `setTimeout` continues to run synchronously

```
console.log('I am printed first');

setTimeout(() => {
  console.log('Printed third!');
}, 2000);

console.log('I am printed second');
```

- `setInterval` is similar to `setTimeout`, but continues to fire the callback on an interval rather than being executed only once
- Like `setTimeout`, `setInterval` returns a value to us so that we can later make reference to the interval (eg. in order to cancel it)

```
// this will log "hello there!" to the console every second until stopped
const interval = setInterval(() => {
  console.log('hello there!');
}, 1000);

// to stop an interval use clearInterval, for timeouts use clearTimeout
clearInterval(interval);
```

```
// stop the interval after 10 iterations
let iterations = 0;

const interval = setInterval(() => {
  iterations++;
  console.log('hello there!');

  if (iterations === 10) {
    clearInterval(interval);
  }
}, 1000);
```

File System Functions

- Node has a built-in module that allows us to interact with the filesystem (ie. read/write to files)
- We can require the `fs` module into our code just like any other module

```
const fs = require('fs');

// will read my-doc.txt synchronously
const data = fs.readFileSync('./my-doc.txt', { encoding: 'utf8' });
console.log(data);

// will read my-doc.txt asynchronously
fs.readFile('./my-doc.txt', { encoding: 'utf8' }, (err, data) => {
  console.log(data);
});
```

Useful Links

- MDN Async Concepts
- Node Docs: Blocking vs Non-Blocking
- Node Docs: fs

View on Compass | Leave Feedback