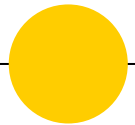
CS4220 Node.js & Angular.js

Cydney Auman
Albert Cervantes
CSULA



Advanced Javascript

Callbacks

Callbacks are functions that are passed as arguments to other functions. This is an important feature of asynchronous programming. It enables the function that receives the callback to call our code when it finishes a long running task, while allowing us to continue the execution of other code.

Simply put a callback is a function to be executed after another function is done executing.

Callbacks

```
const oddEven = (n, callback) => {
    if (n % 2 === 0)
        callback(null, n)
    else
        callback(true, n)
}

oddEven(2, (err, result) => {
    if (err)
        console.log('is odd: ' + result)
    else
        console.log('is even: ' + result)
})
```

Promises

A Promise is an object which takes a callback. A promise specifies some code to be executed later (as with callbacks) and also explicitly indicates whether the code succeeded or failed at its job. You can chain promises together based on success or failure.

A **Promise** is in one of these states:

- *pending*: initial state, not fulfilled or rejected.
- *fulfilled*: meaning that the operation completed successfully.
- *rejected*: meaning that the operation failed.

- The `Promise.all(iterable)` method returns a promise that resolves when all of the promises in the iterable argument have resolved, or rejects with the reason of the first passed promise that rejects.

Promises

```
const oddEven = (n) => {
    return new Promise((resolve, reject) => {
        if (n % 2 === 0)
            resolve(n)
        else
            reject(n)
    })
}

oddEven(2)
    .then((result) => {
        console.log('is even: ' + result)
    })
    .catch((result) => {
        console.log('is odd: ' + result)
    })
```

Event Loop

JavaScript engines are built on the concept of a single-threaded event loop (one piece of code is ever executed at a time). Java or C++ can allow multiple different pieces of code to execute at the same time.

The event loop is a process inside the JavaScript engine that monitors code execution and manages the jobs.

https://www.youtube.com/watch?v=8aGhZQkoFbQ

Timers

`setTimeout()`

Calls a function or executes a code after specified delay.

`setInterval()`

Calls a function or executes a code repeatedly, with a fixed time delay between each call to that function.

`clearInterval()`

Cancels repeated action which was set up using setInterval().

References & Readings

Asynchronous Programming, Promises, Callback, & Event Loop

https://leanpub.com/understandinges6/read#leanpub-auto-promises-and-asynchronous-programming

Promises

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Event Loop

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

Timers

https://developer.mozilla.org/en-US/Add-ons/Code_snippets/Timers