Modern Asynchronous Programming with JavaScript (Goodbye Callbacks!)

Henrique S. Coelho
Matthew Welke

The Message Queue and Event Loop

Code is a Series of Instructions

- Computer reads your code one line at a time.
- Before one line can be executed, the line before it must finish executing.
- We call this “synchronous” programming.

```
const x = 42;
const answer = x;
console.log(`The answer to life is ${answer}`);
```

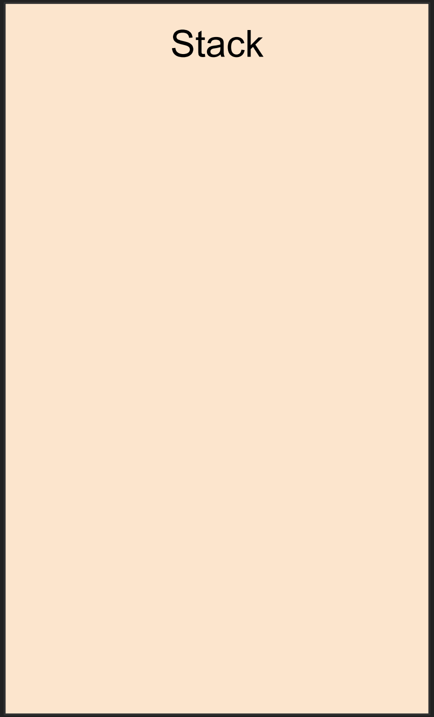
Code is a ~~Series of Instructions~~ Giant While Loop

- The CPU runs at a certain frequency.
- 2.2 GHz = 2,200,000,000 clock cycles per second.
- One clock cycle is used to perform a tiny operation.
- Those tiny operations add up to doing things like "make a new variable called x and put 42 into it".

What does this mean?

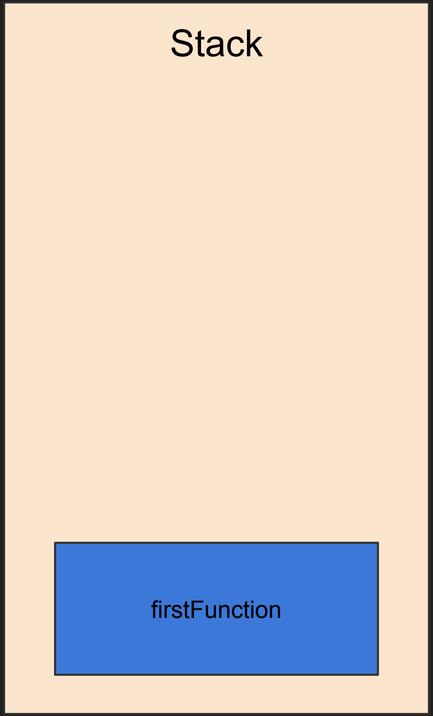
The Computer could potentially do billions of "things" per second.

Normal Programming

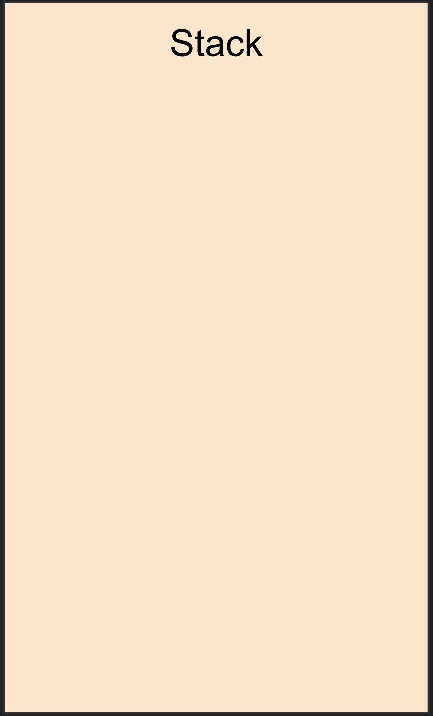


What is the stack?

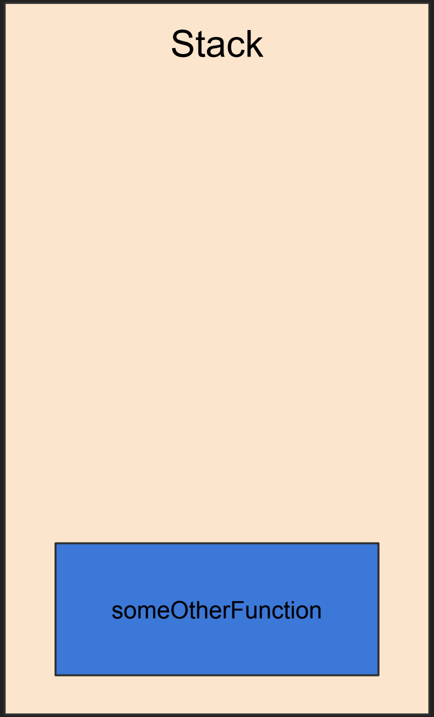
Normal Programming



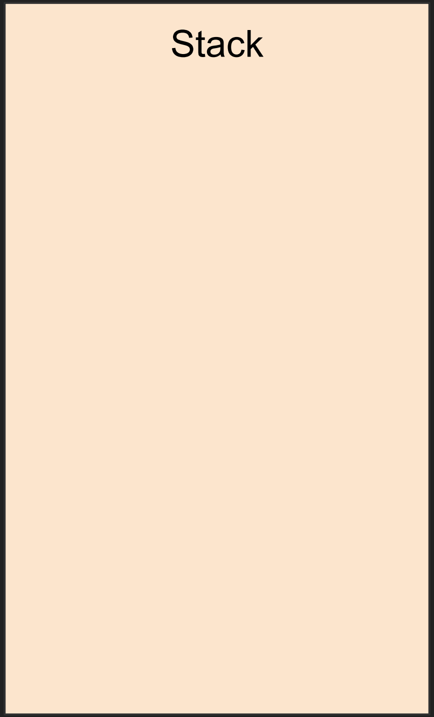
Normal Programming



Normal Programming

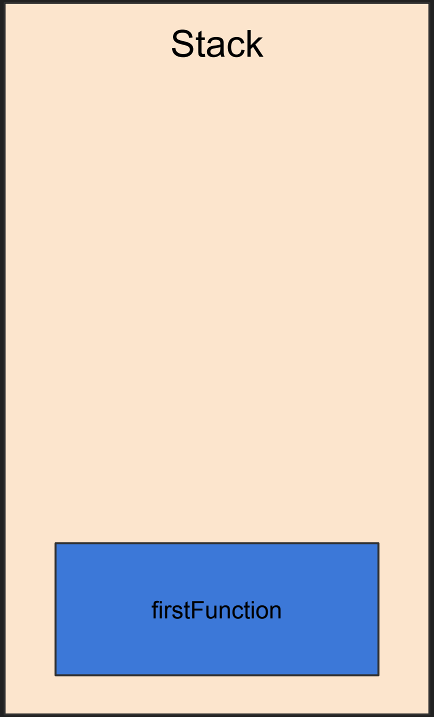


Normal Programming



Stack

Normal Programming

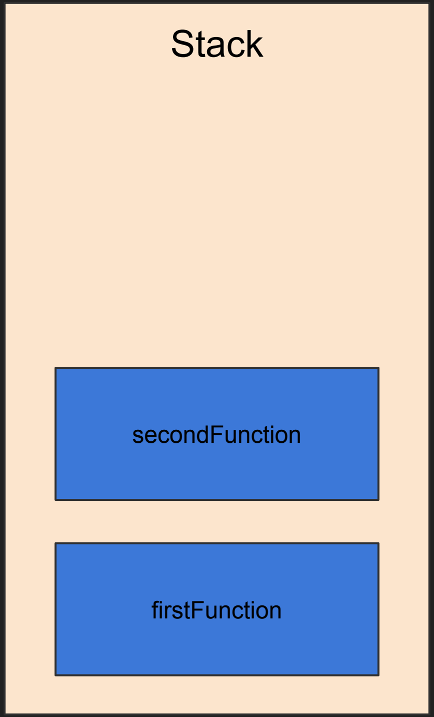


Stack

firstFunction

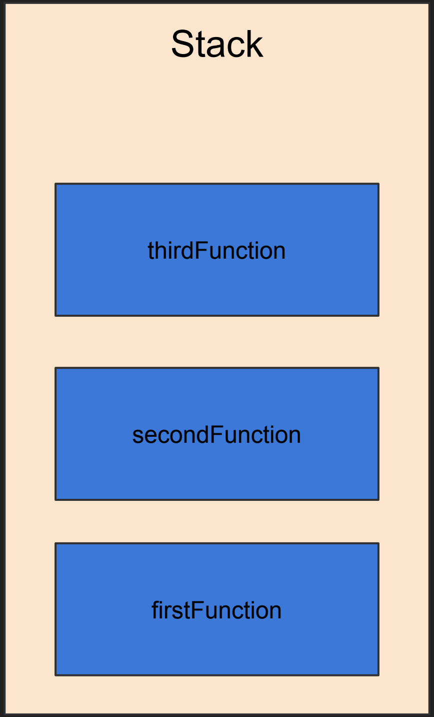
```
function firstFunction() {
  secondFunction();
}
```

Normal Programming



```
function firstFunction() {
  secondFunction();
}
```

Normal Programming



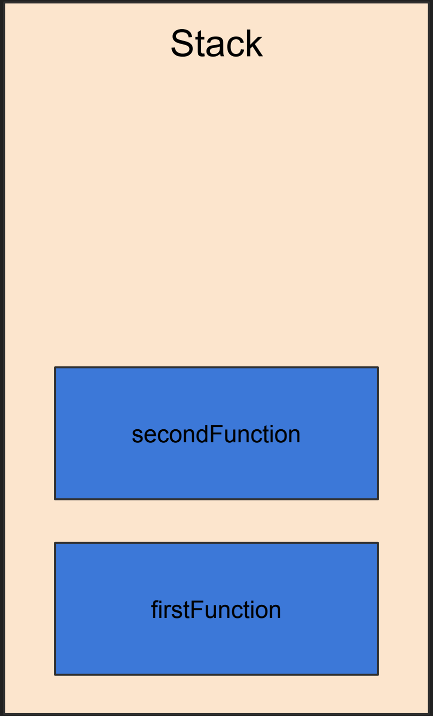
Stack

thirdFunction

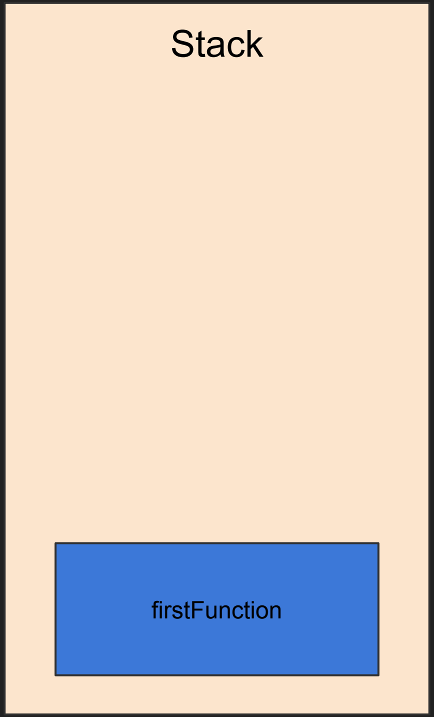
secondFunction

firstFunction

Normal Programming



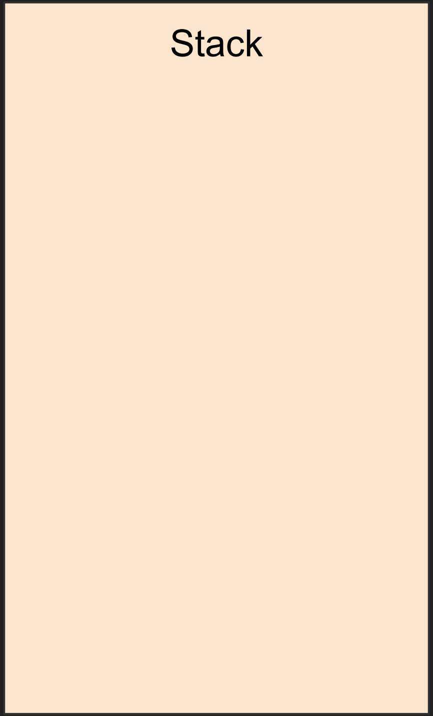
Normal Programming



Stack

firstFunction

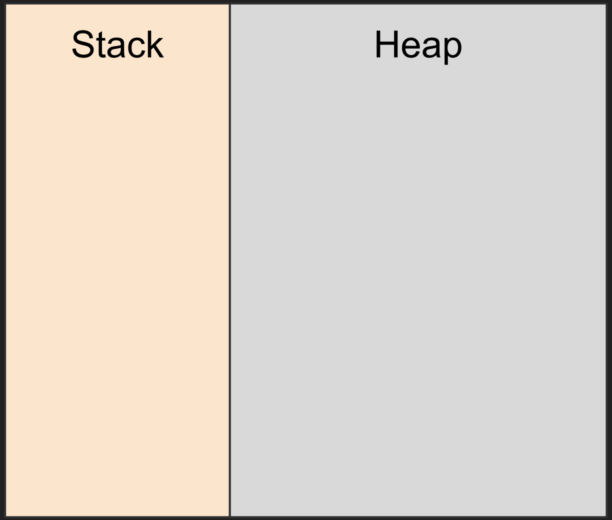
Normal Programming



Stack

Normal Programming

```
function cleanKitchen() {
  console.log(‘Cleaned kitchen.’);
}

function cleanHome() {
  cleanKitchen();
  console.log(‘Cleaned home.’);
}

cleanHome();
```

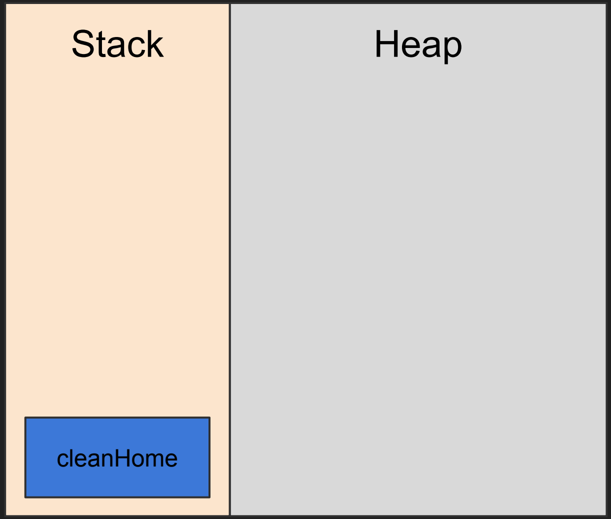


Stack

Heap

Normal Programming

```
function cleanKitchen() {
  console.log(‘Cleaned kitchen.’);
}

function cleanHome() {
  cleanKitchen();
  console.log(‘Cleaned home.’);
}

cleanHome();
```



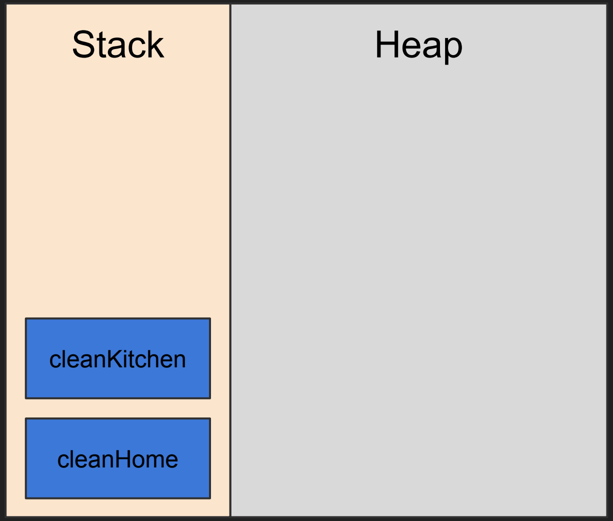
Stack

Heap

cleanHome

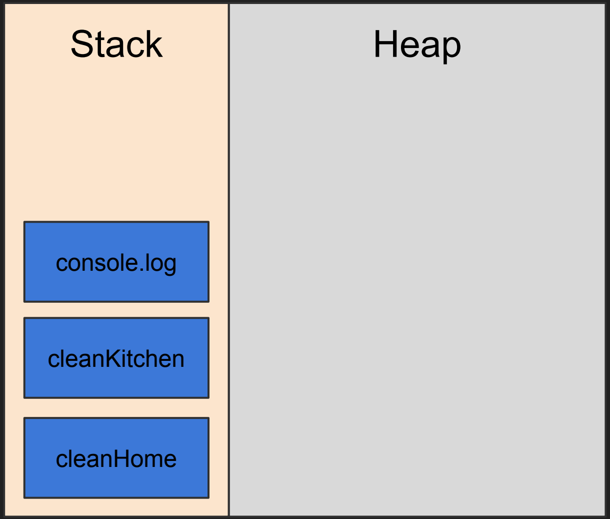
Normal Programming

```
function cleanKitchen() {
  console.log(‘Cleaned kitchen.’);
}

function cleanHome() {
  cleanKitchen();
  console.log(‘Cleaned home.’);
}

cleanHome();
```



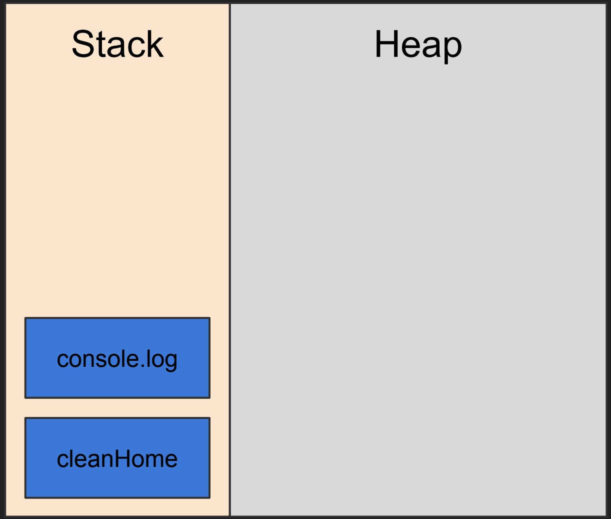
Normal Programming

```
function cleanKitchen() {
  console.log(‘Cleaned kitchen.’);
}

function cleanHome() {
  cleanKitchen();
  console.log(‘Cleaned home.’);
}

cleanHome();
```



Normal Programming

```
function cleanKitchen() {
  console.log(‘Cleaned kitchen.’);
}

function cleanHome() {
  cleanKitchen();
  console.log(‘Cleaned home.’);
}

cleanHome();
```



Stack

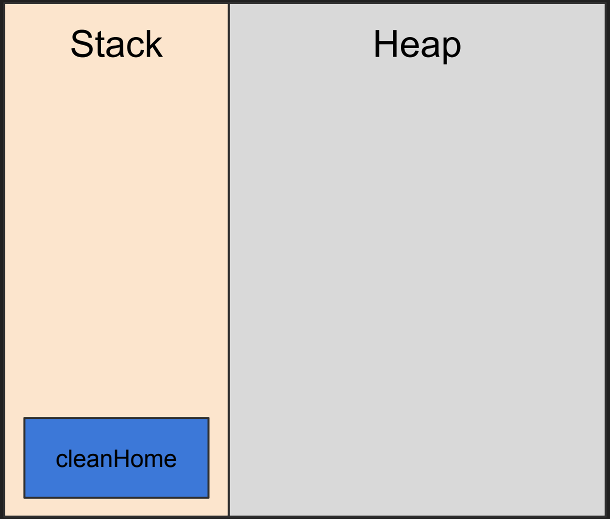
Heap

console.log

cleanHome

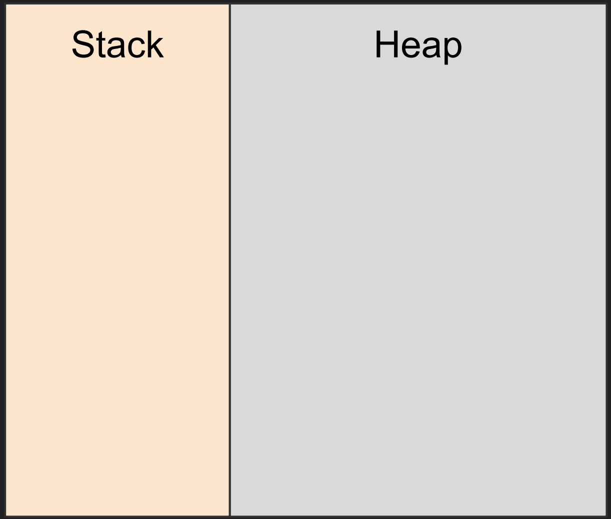
Normal Programming

```
  function cleanKitchen() {
    console.log(‘Cleaned kitchen.’);
  }

  function cleanHome() {
    cleanKitchen();
    console.log(‘Cleaned home.’);
  }

→ cleanHome();
```



Normal Programming

```
function cleanKitchen() {
  console.log(‘Cleaned kitchen.’);
}

function cleanHome() {
  cleanKitchen();
  console.log(‘Cleaned home.’);
}

cleanHome();
```

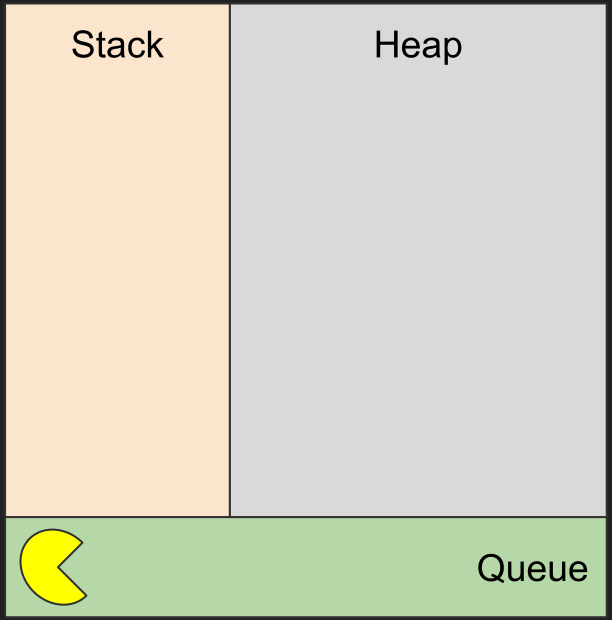


Stack

Heap

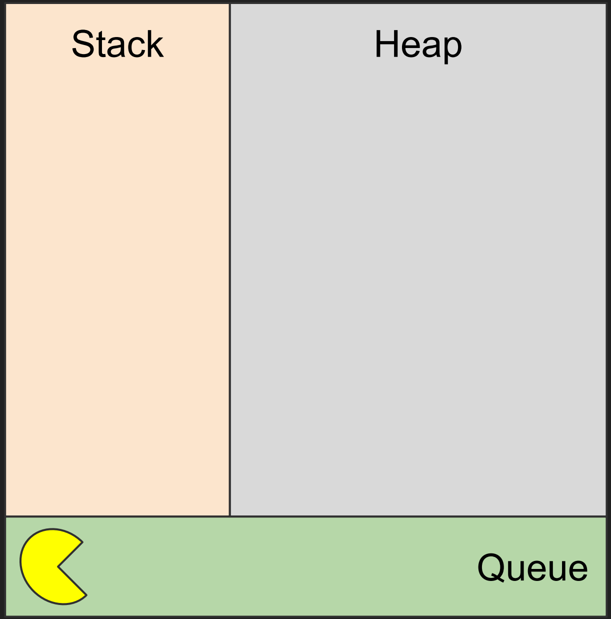
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



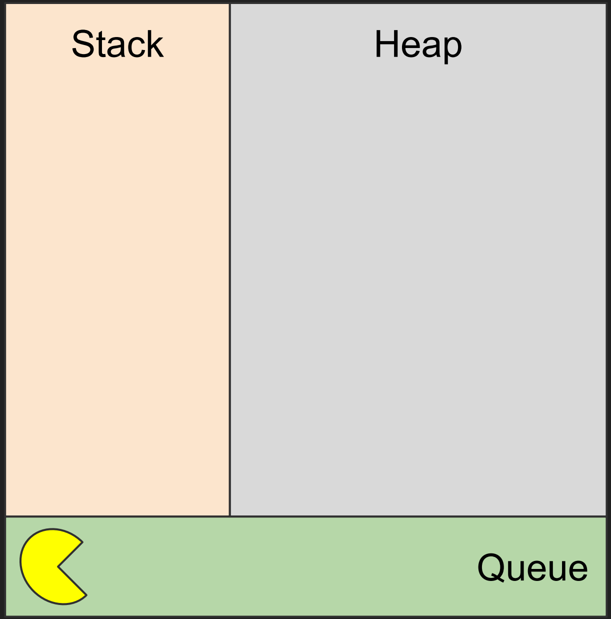
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



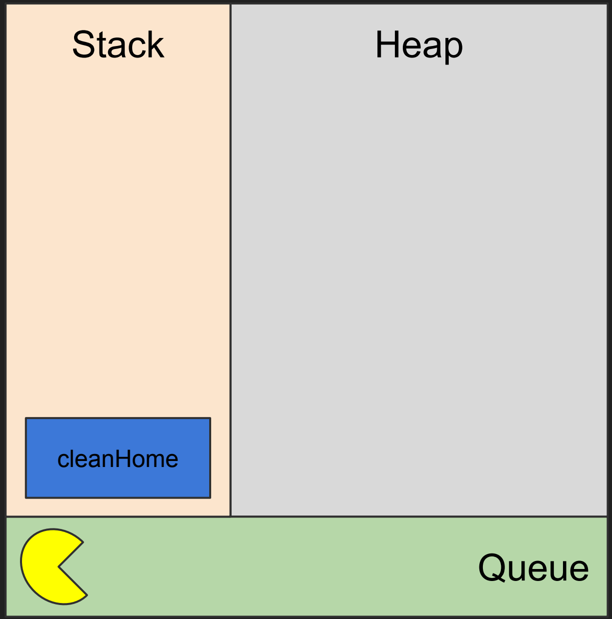
Stack

Heap

Queue

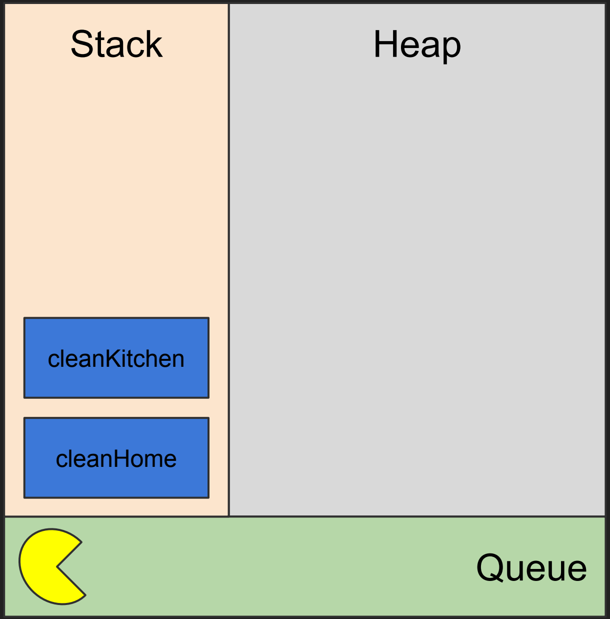
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



Stack

Heap

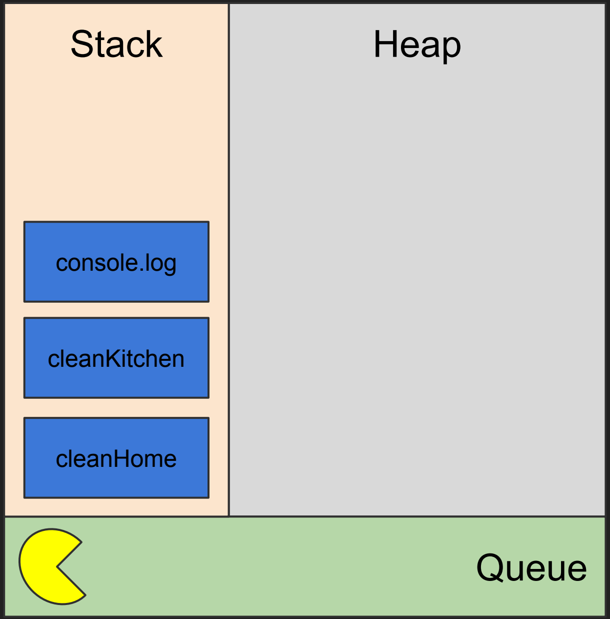
cleanKitchen

cleanHome

Queue

Asynchronous JavaScript Programming

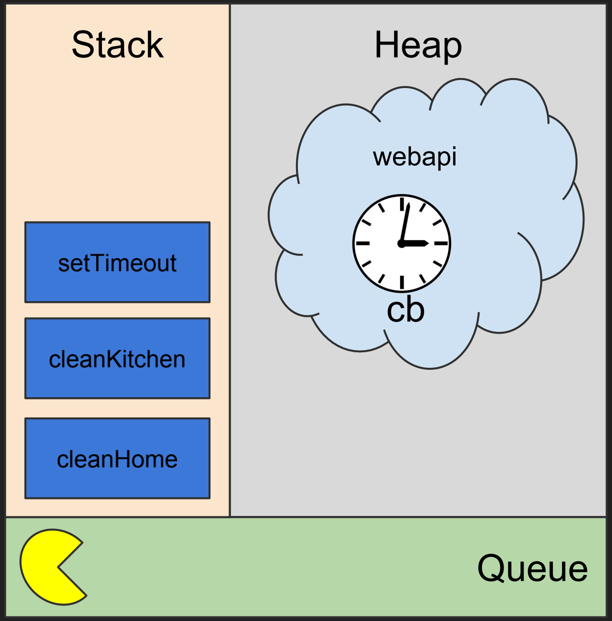
```
function cleanKitchen() {
  console.log('Started dishwasher.');
  setTimeout(function cb() {
    console.log('Dishwasher done. Cleaned kitchen.');
    console.log('Cleaned home.');
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



Asynchronous JavaScript Programming

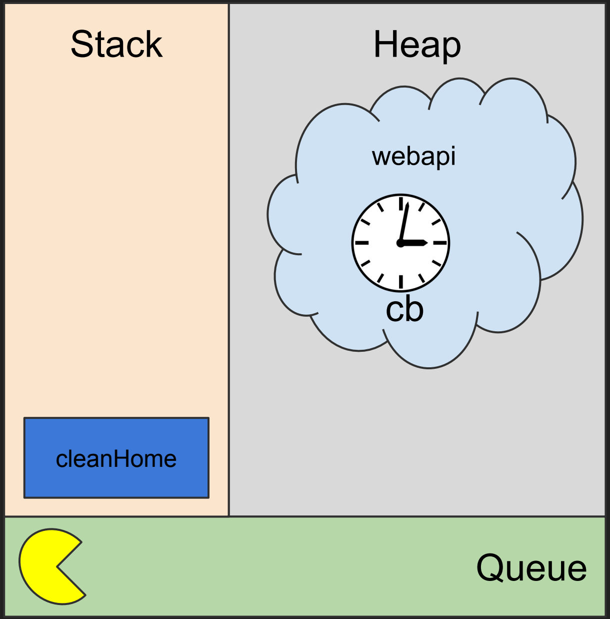
```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```

- You wouldn’t just stand by the dishwasher and wait for it to finish.
- You’d start it and work on something else.



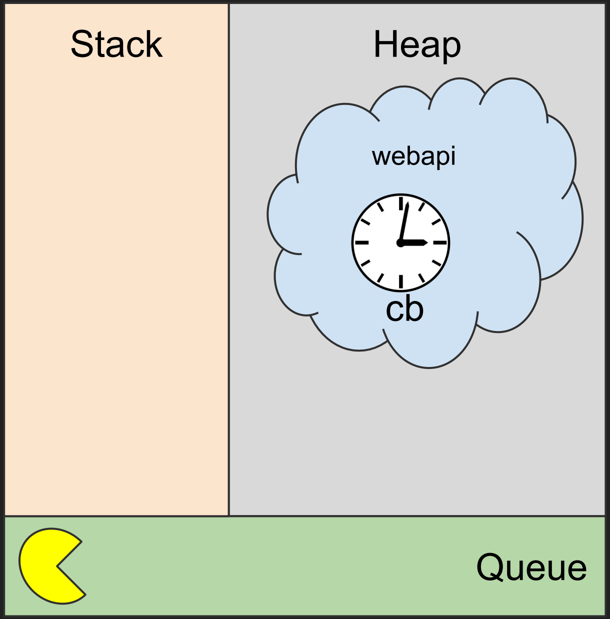
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log('Started dishwasher.');
  setTimeout(function cb() {
    console.log('Dishwasher done. Cleaned kitchen.');
    console.log('Cleaned home.');
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



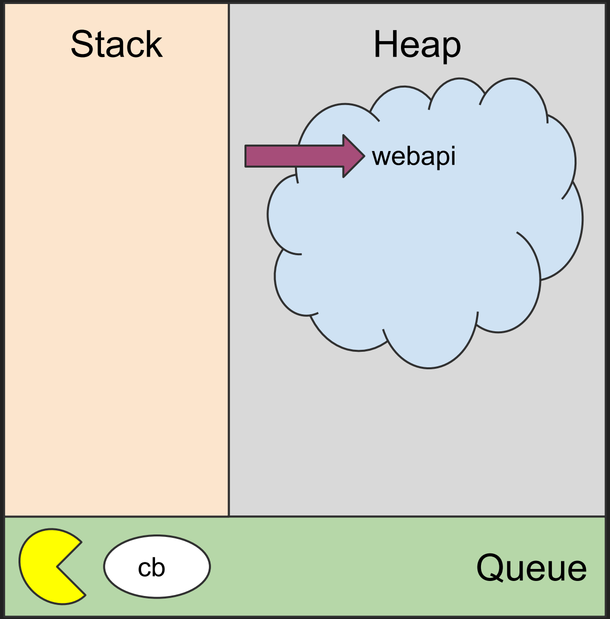
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log('Started dishwasher.');
  setTimeout(function cb() {
    console.log('Dishwasher done. Cleaned kitchen.');
    console.log('Cleaned home.');
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



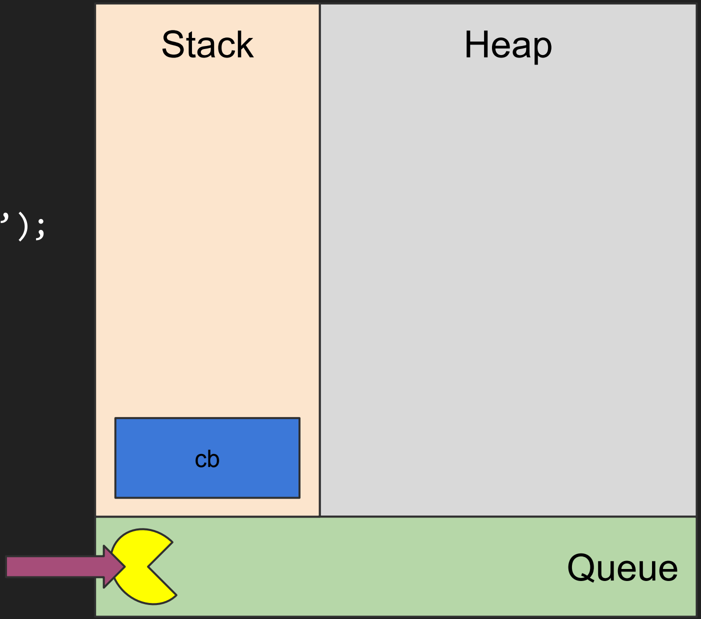
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



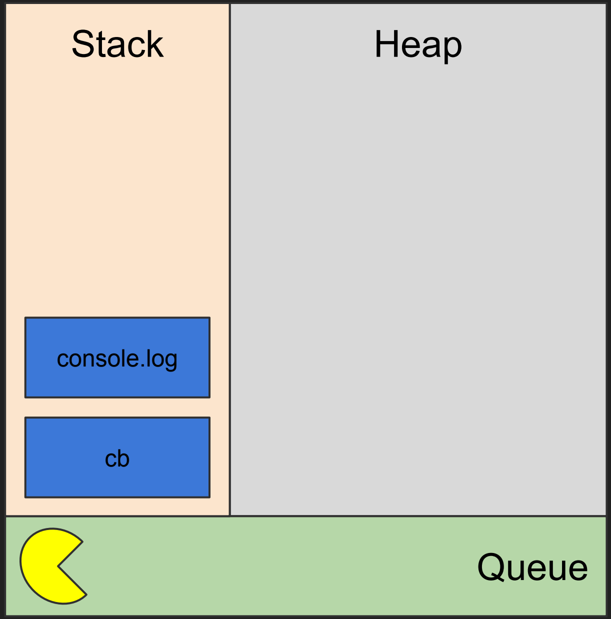
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



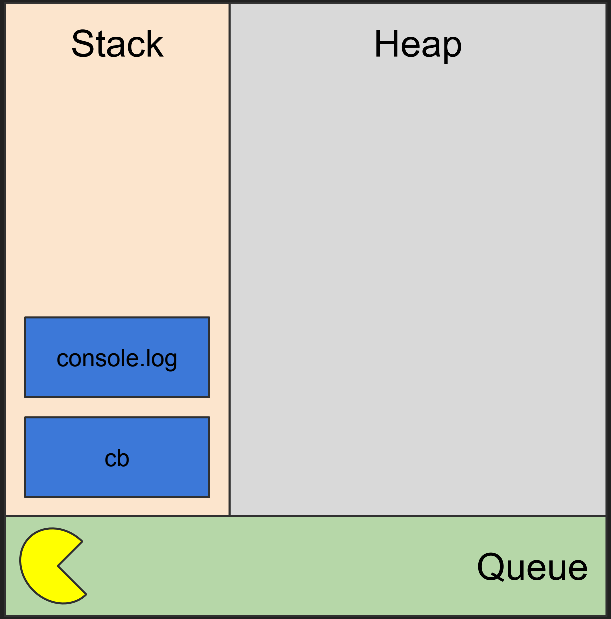
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



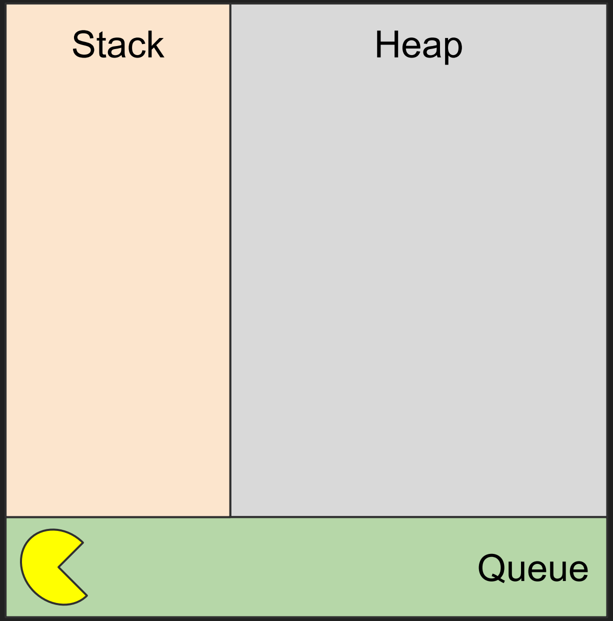
Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



Asynchronous JavaScript Programming

```
function cleanKitchen() {
  console.log(‘Started dishwasher.’);
  setTimeout(function cb() {
    console.log(‘Dishwasher done. Cleaned kitchen.’);
    console.log(‘Cleaned home.’);
  }, 30000); // Add msg to queue in 30s
}

function cleanHome() {
  cleanKitchen();
}

cleanHome();
```



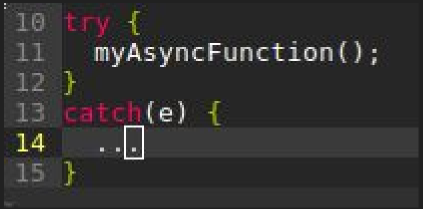
Some things have to be done in synchrony

> getInformationFromDatabase();

> buildHTMLTemplate();

> sendResponse();

No try/catch



```
try {
  myAsyncFunction();
}
catch(e) {
  ...
}
```

Does not work!

example 1.js

Methods of Asynchronous programming

1. Callbacks

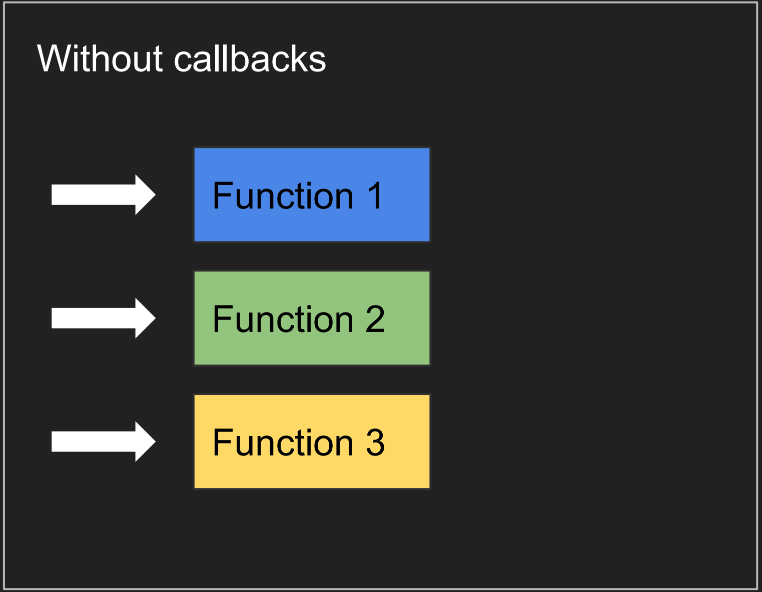
2. Promises (ES6)

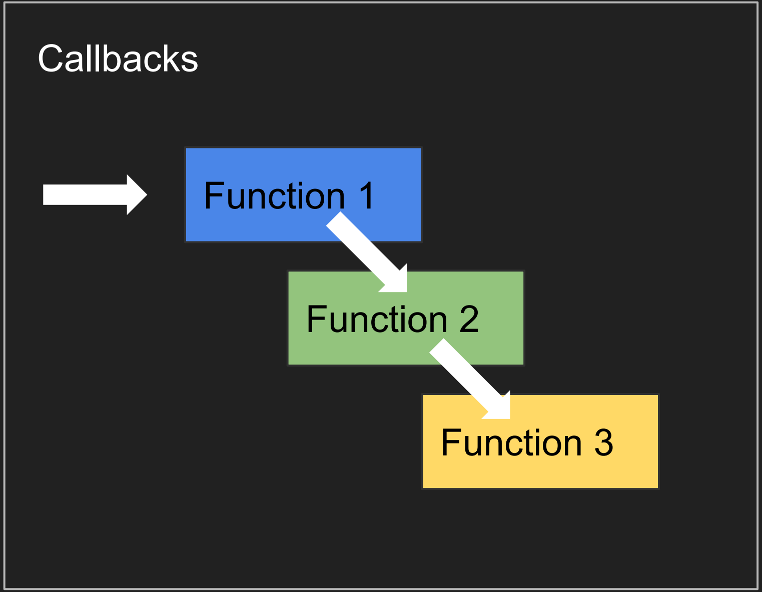
3. Generators (ES6)

4. Async functions (ES7)

Callbacks

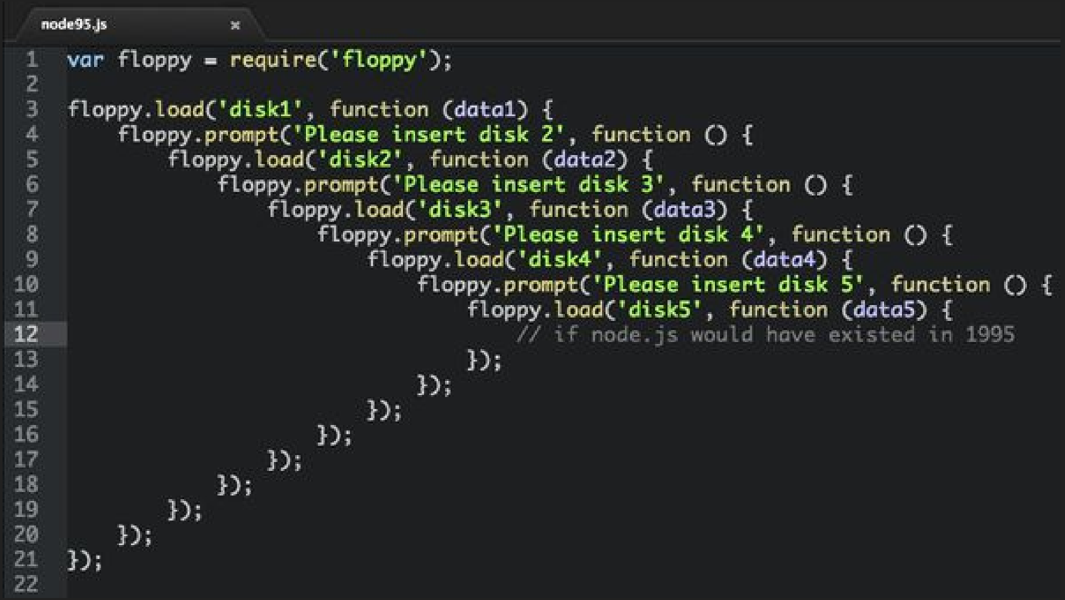
A chain of instructions. Pass what to do next when the function is done



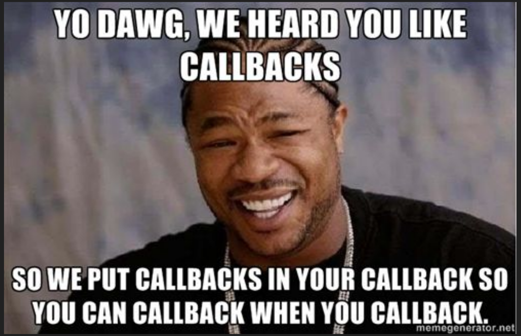


example callback1.js, callback2.js, callback3.js

Callback hell



```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function () {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function () {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function () {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function () {
                                floppy.load('disk5', function (data5) {
                                    // if node.js would have existed in 1995
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});
```

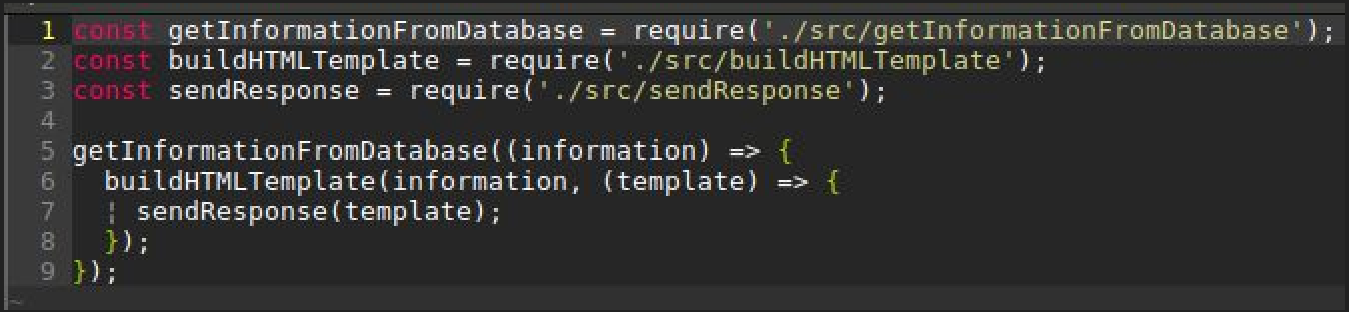


YO DAWG, WE HEARD YOU LIKE CALLBACKS
SO WE PUT CALLBACKS IN YOUR CALLBACK SO YOU CAN CALLBACK WHEN YOU CALLBACK.
memegenerator.net

example callback1.js

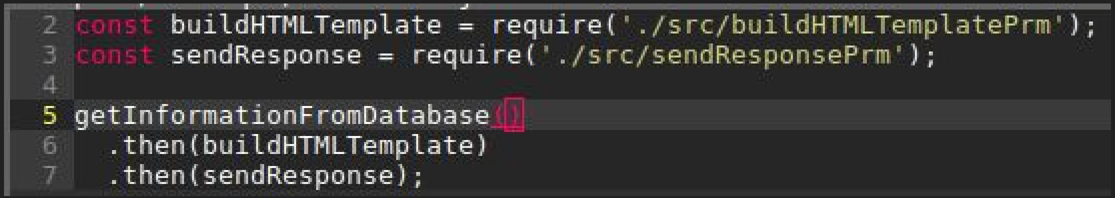
Promises

Callbacks



```
const getInformationFromDatabase = require('./src/getInformationFromDatabase');
const buildHTMLTemplate = require('./src/buildHTMLTemplate');
const sendResponse = require('./src/sendResponse');

getInformationFromDatabase((information) => {
  buildHTMLTemplate(information, (template) => {
    sendResponse(template);
  });
});
```

Promises



```
const buildHTMLTemplate = require('./src/buildHTMLTemplatePrm');
const sendResponse = require('./src/sendResponsePrm');

getInformationFromDatabase()
  .then(buildHTMLTemplate)
  .then(sendResponse);
```

example promise1.js, promise2.js

Promises

Other useful methods:

Promise.all([Promises]) - Succeeds as soon as one of the promises finish

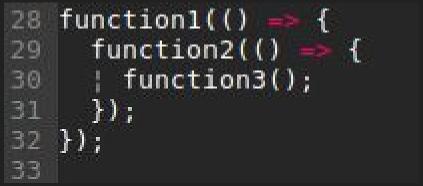
Promise.race([Promises]) - Succeeds when all the promises succeed

example promise3.js

Generators

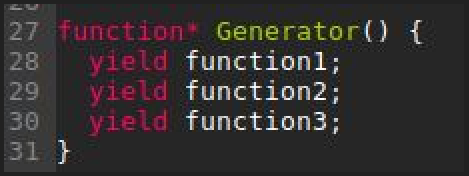
In simple terms, it is a factory for iterators

Iterators are collections that allow you to access its content one item at a time, normally with a .next() method



```
function1(() => {
  function2(() => {
    function3();
  });
});
```

→



```
function* Generator() {
  yield function1;
  yield function2;
  yield function3;
}
```

example generator1.js

Async Functions

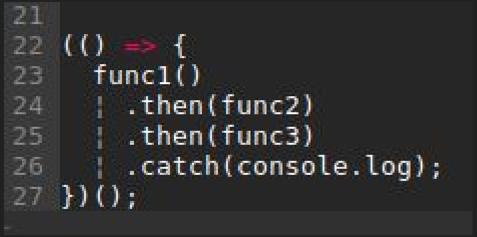
New in ES7

Functions that return promises (have the method .then())

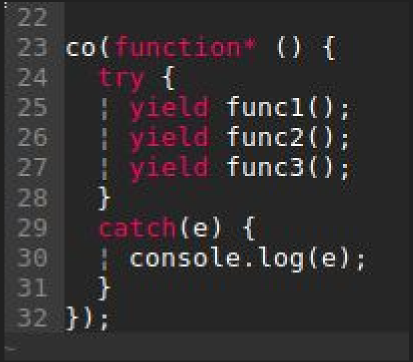
Provide the keyword await - it waits for promises to be completed (somehow similar to yield)

example async1.js, async2.js

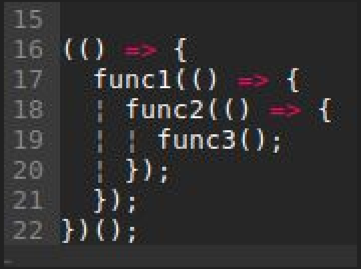
Conclusion



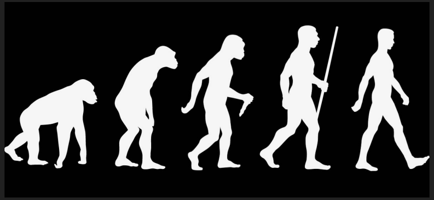
```
(() => {
  func1()
    .then(func2)
    .then(func3)
    .catch(console.log);
})();
```

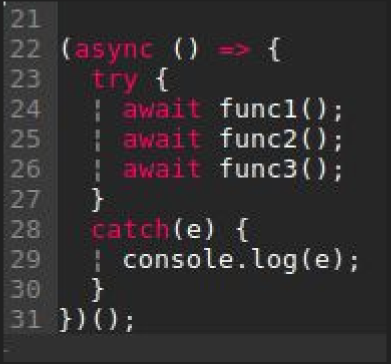


```

co(function* () {
  try {
    yield func1();
    yield func2();
    yield func3();
  }
  catch(e) {
    console.log(e);
  }
});
```



```

(() => {
  func1(() => {
    func2(() => {
      func3();
    });
  });
})();
```





```

(async () => {
  try {
    await func1();
    await func2();
    await func3();
  }
  catch(e) {
    console.log(e);
  }
})();
```