Event Looping

JavaScript has a concurrency model based on an **event loop**, which is responsible for executing the code, collecting and processing events, and executing queued sub-tasks. This model is quite different from models in other languages like C and Java.

Blocking the event loop:

Any JavaScript code that takes too long to return back control to the event loop will block the execution of any JavaScript code in the page, even block the UI thread, and the user cannot click around, scroll the page, and so on.

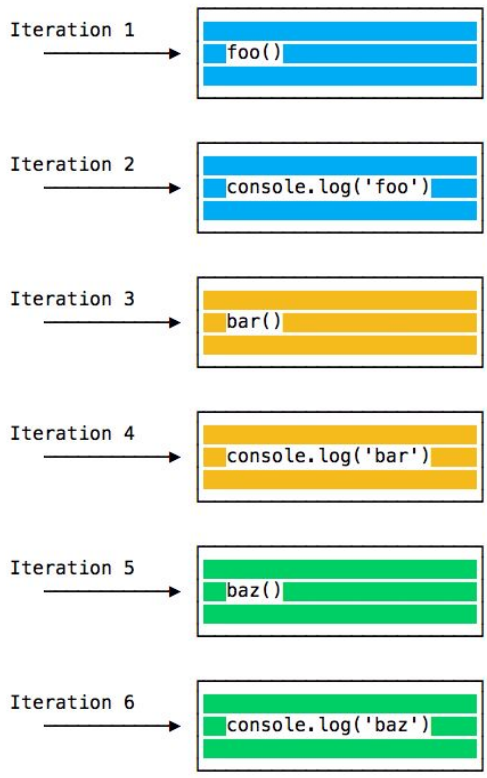
Almost all the I/O primitives in JavaScript are non-blocking. Network requests, Node.js filesystem operations, and so on. Being blocking is the exception, and this is why JavaScript is based so much on callbacks, and more recently on promises and async/await.

Event loop example

```
1. const bar = () => console.log('bar')
2. const baz = () => console.log('baz')
3. const foo = () => {
4.   console.log('foo')
5.   bar()
6.   baz()
7. }
8. foo()
```

Output:

```
foo
bar
baz
```

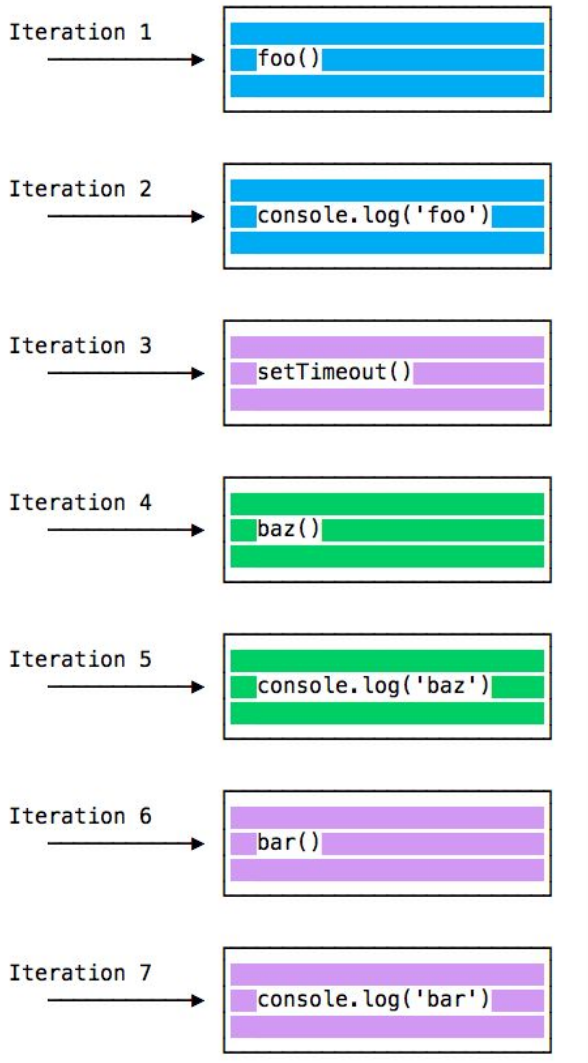


Queuing function execution

The above example looks normal, there's nothing special about it: JavaScript finds things to execute, runs them in order.Let's see how to defer a function until the stack is clear.The use case of `setTimeout(() => {}), 0)` is to call a function, but execute it once every other function in the code has executed.

```
const bar = () => console.log('bar')
```

```
const baz = () => console.log('baz')
const foo = () => {
  console.log('foo')
  setTimeout(bar, 0)
  baz()
}
foo()
```



Callback Hell

Callback hell is any code where the use of function **callbacks** in async code becomes obscure or difficult to follow. Generally, when there is more than one level of indirection, code using **callbacks** can become harder to follow, harder to refactor, and harder to test.

It occurs because in JavaScript the only way to delay a computation so that it runs after the asynchronous call returns is to put the delayed code inside a callback function. You cannot delay code that was written in traditional synchronous style so you end up with nested callbacks everywhere.