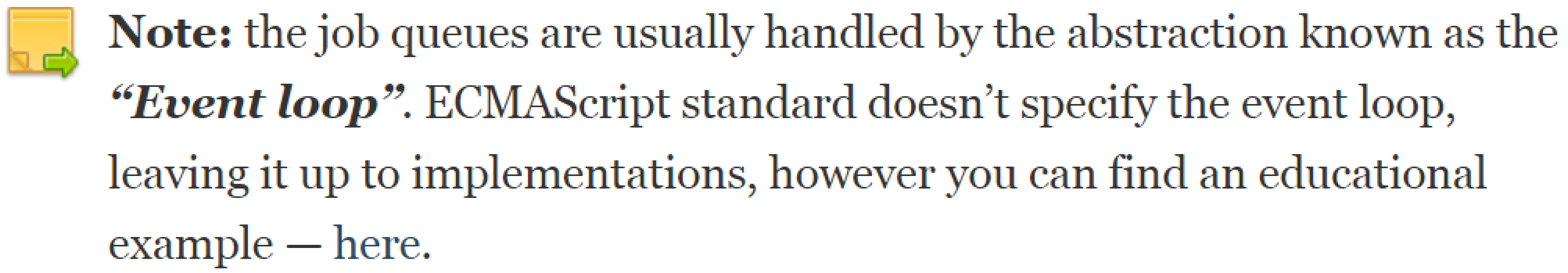
JS → p.6

Asynchronní programování

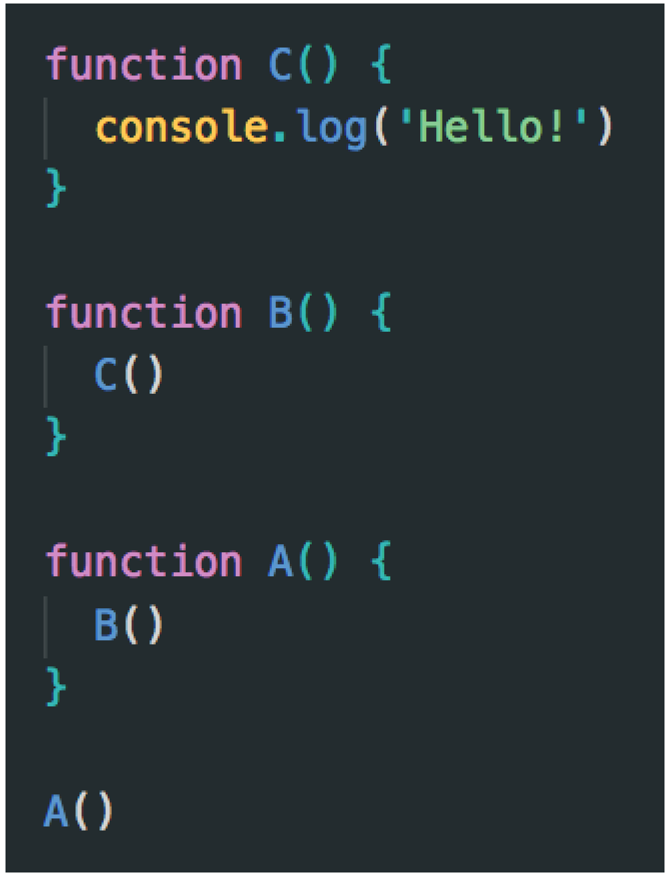
event-loop, neblokující kód

Event-loop

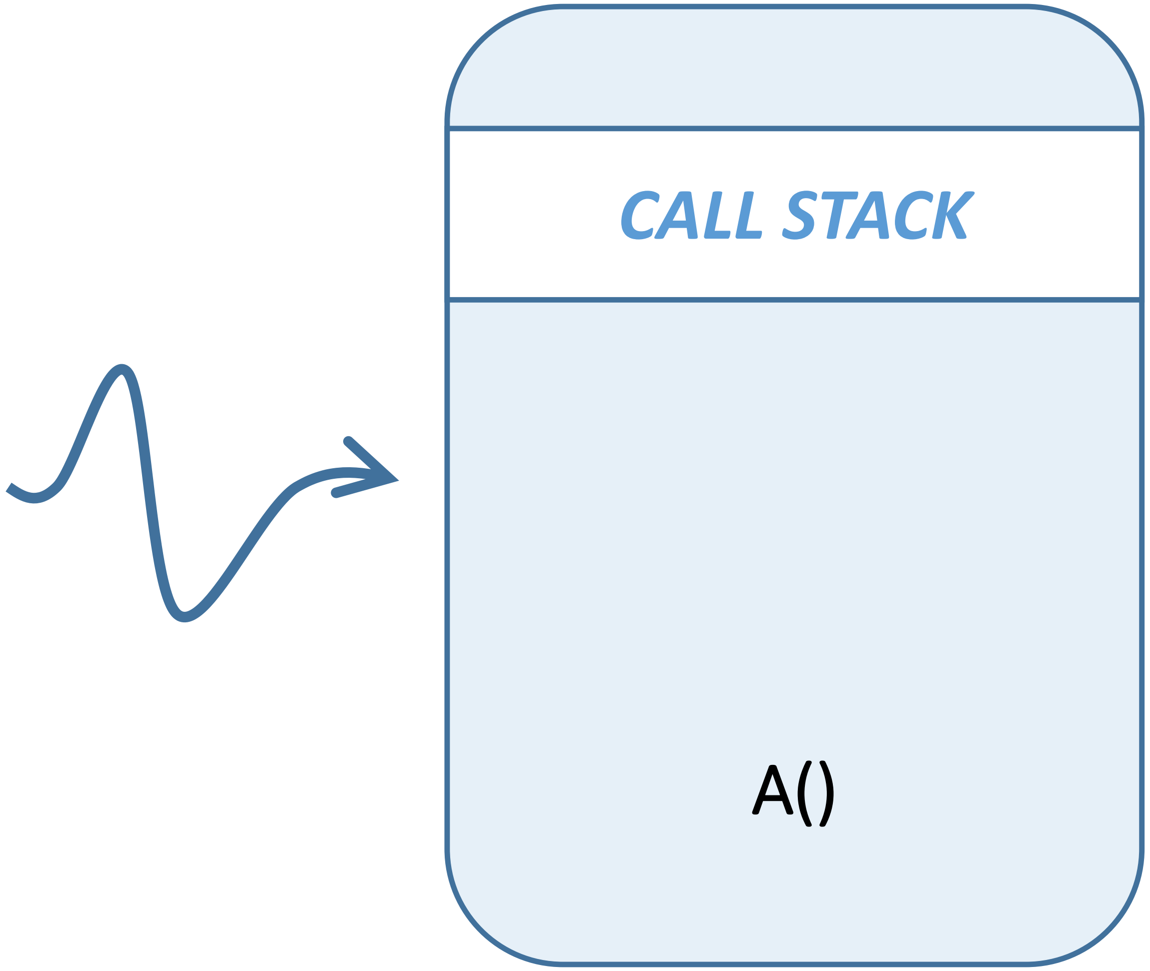


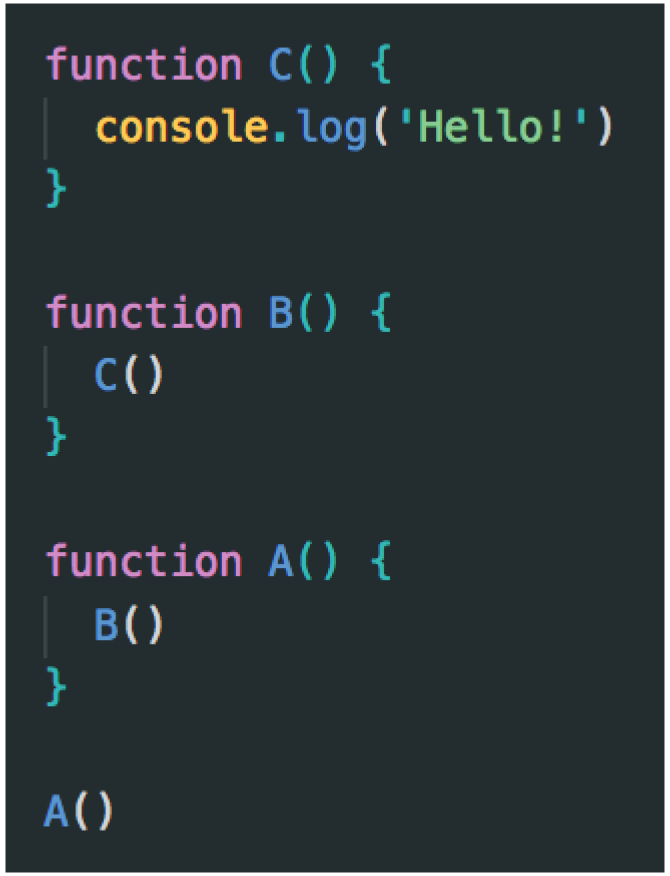
Note: the job queues are usually handled by the abstraction known as the ***"Event loop"***. ECMAScript standard doesn't specify the event loop, leaving it up to implementations, however you can find an educational example — here.

http://dmitrysoshnikov.com/ecmascript/javascript-the-core-2nd-edition/#job

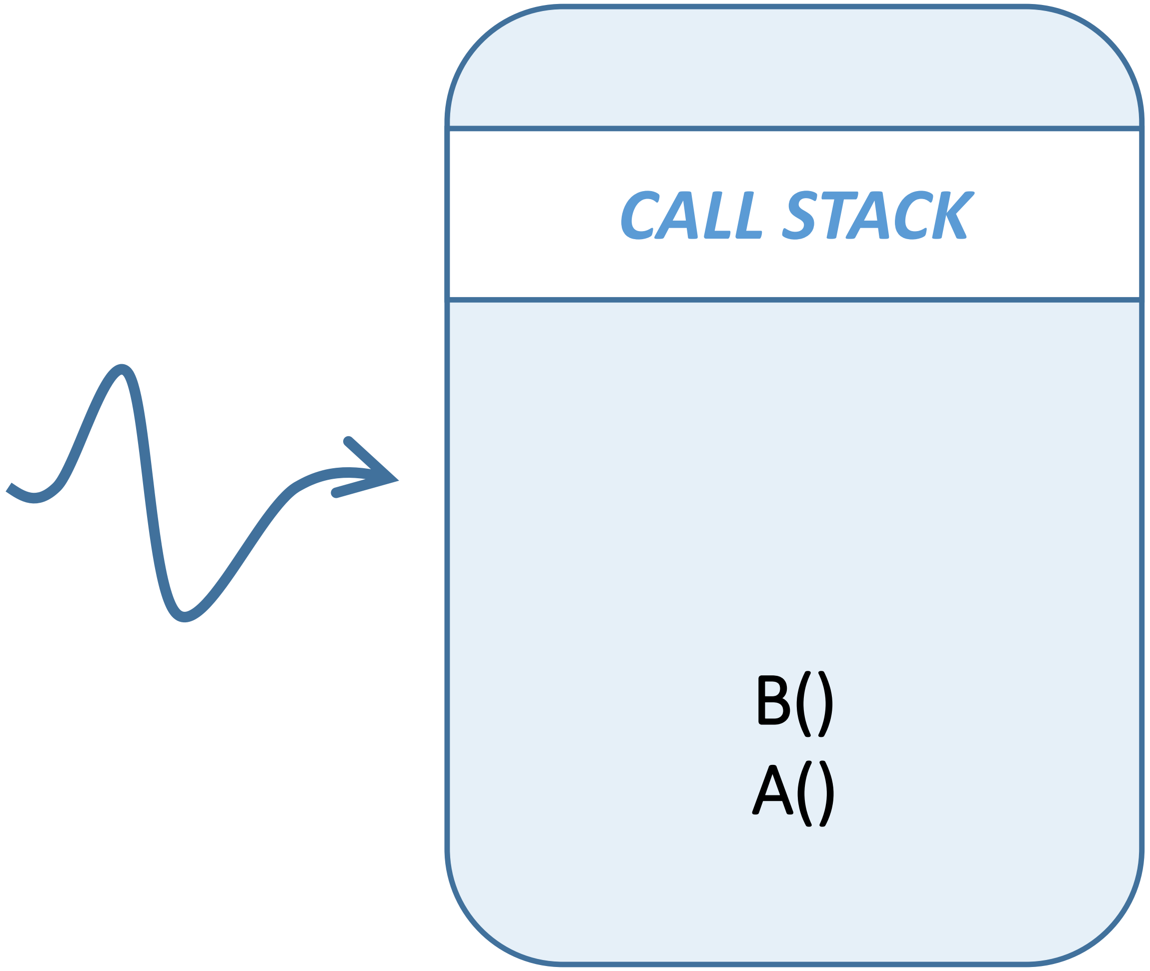


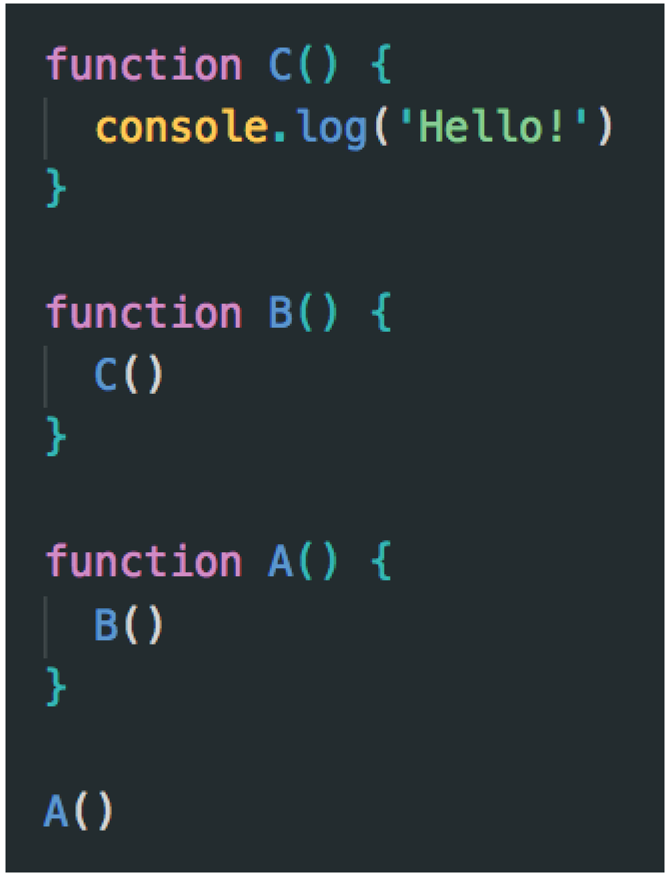
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



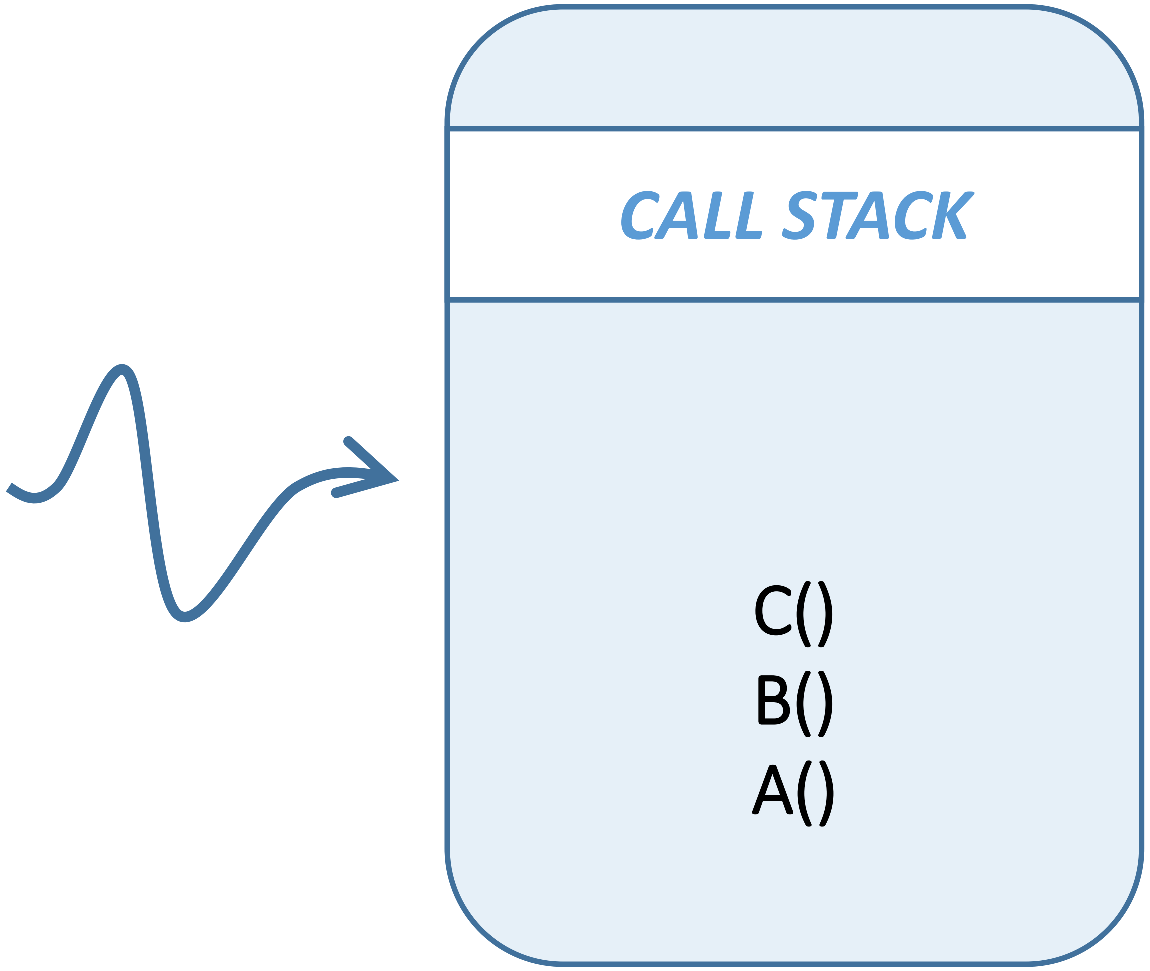


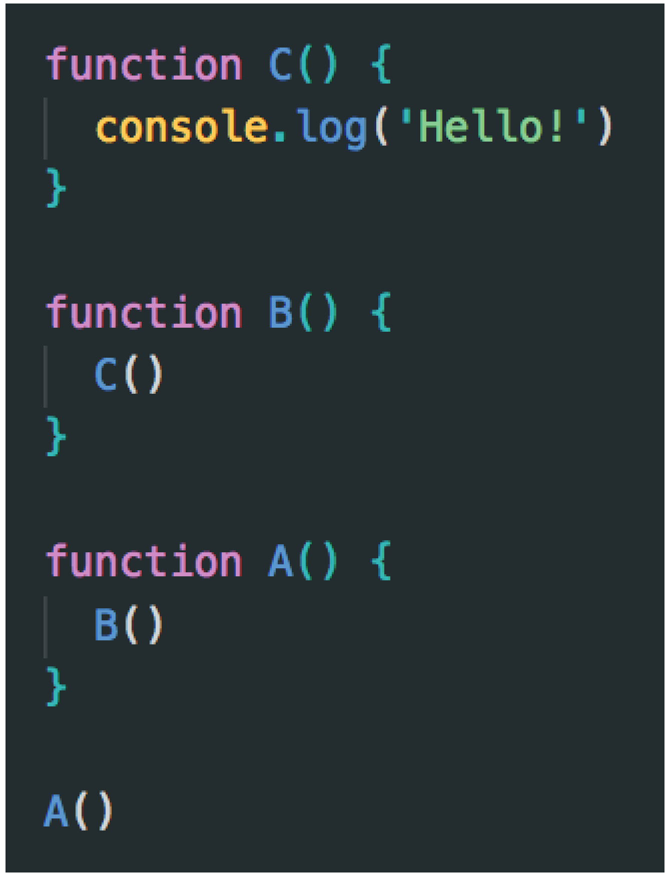
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



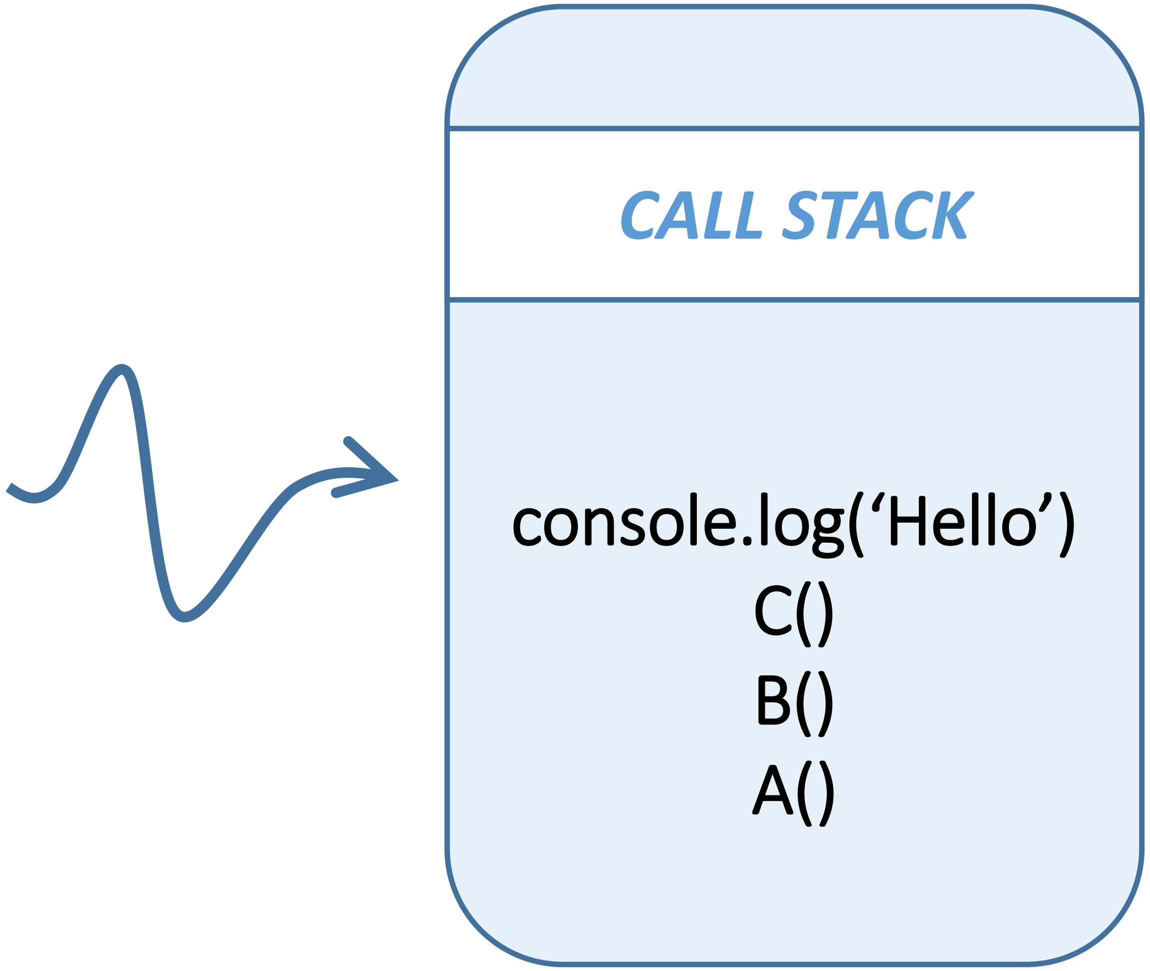


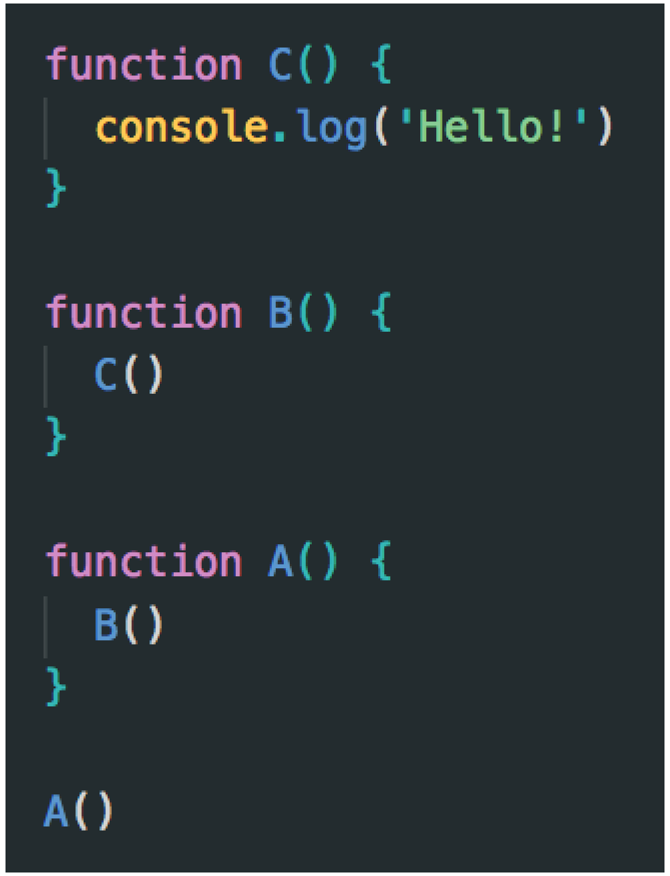
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



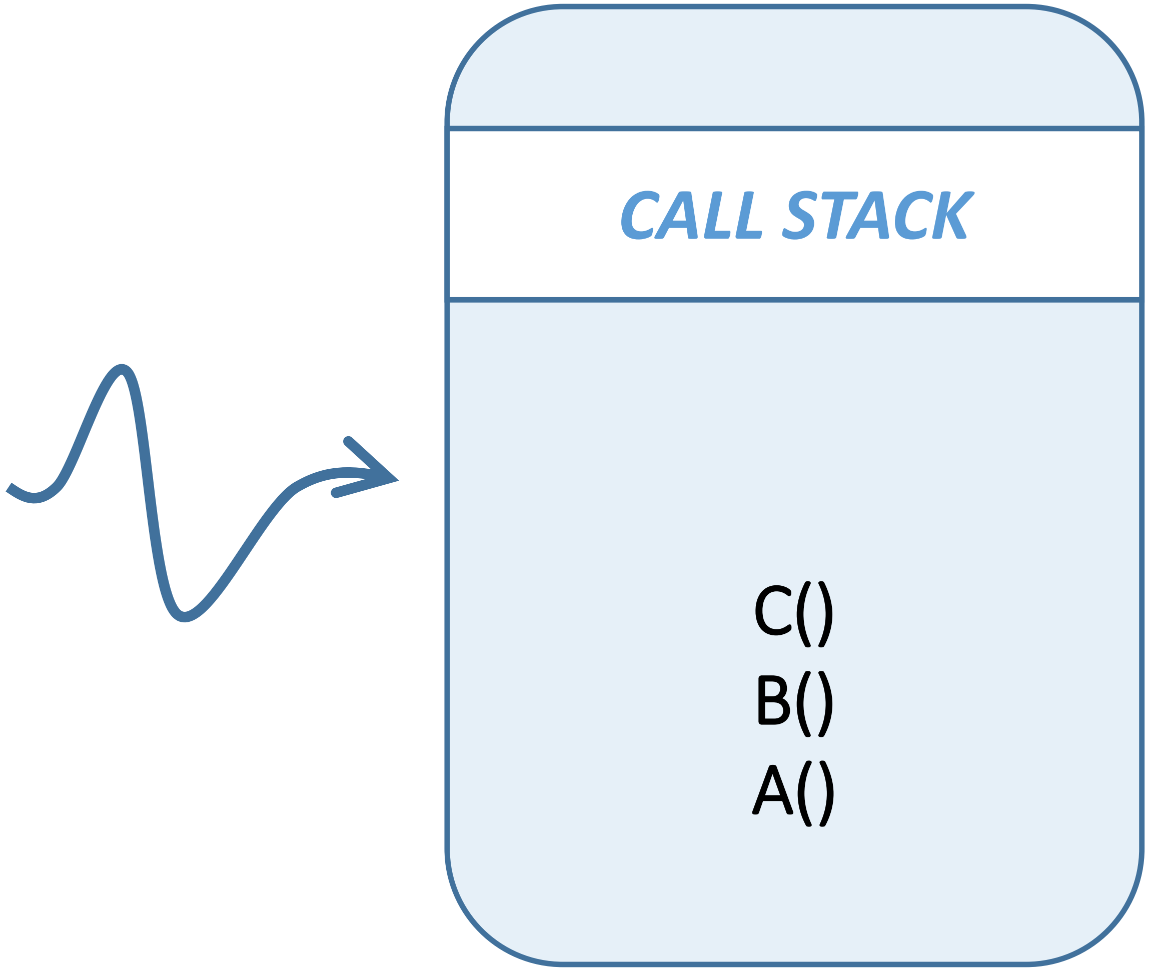


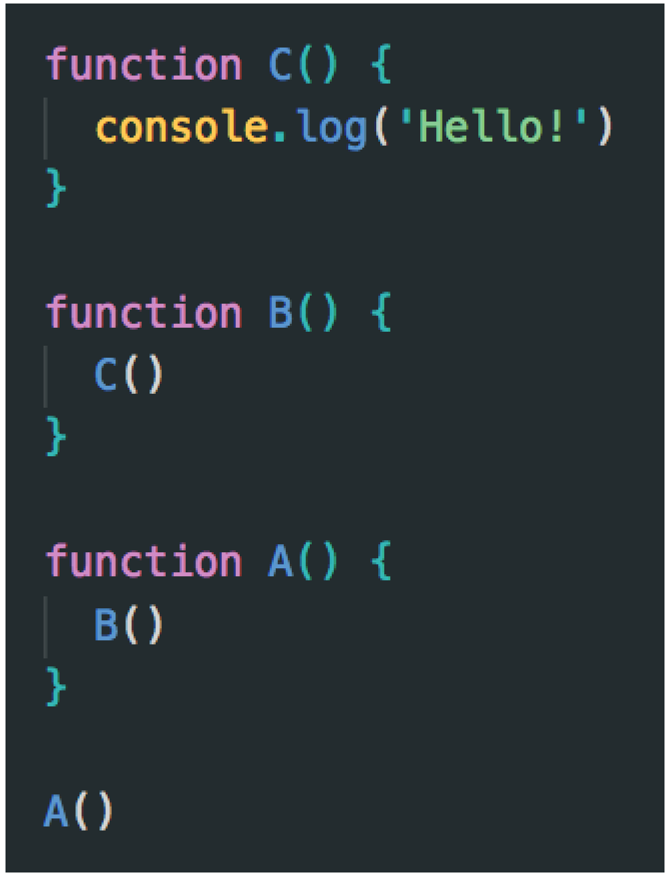
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



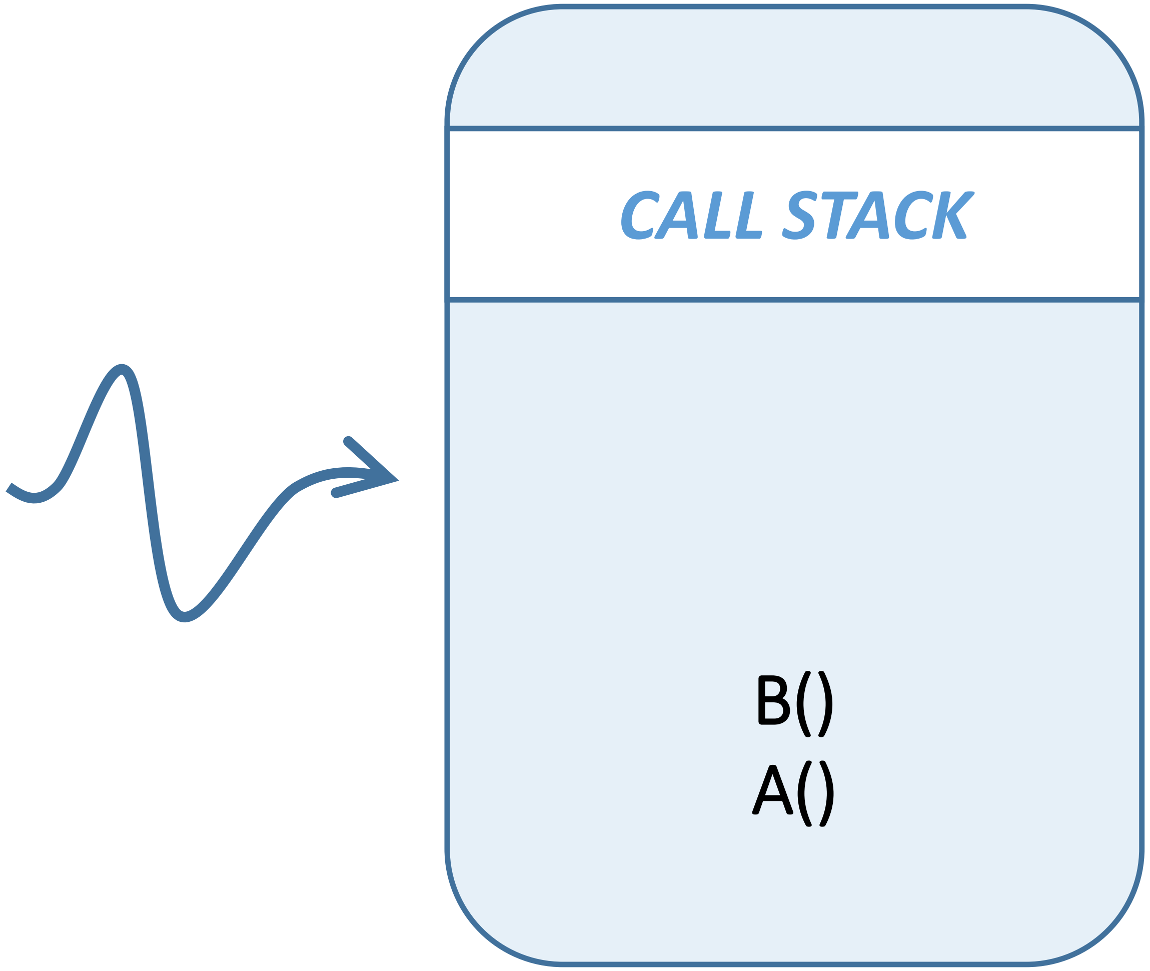


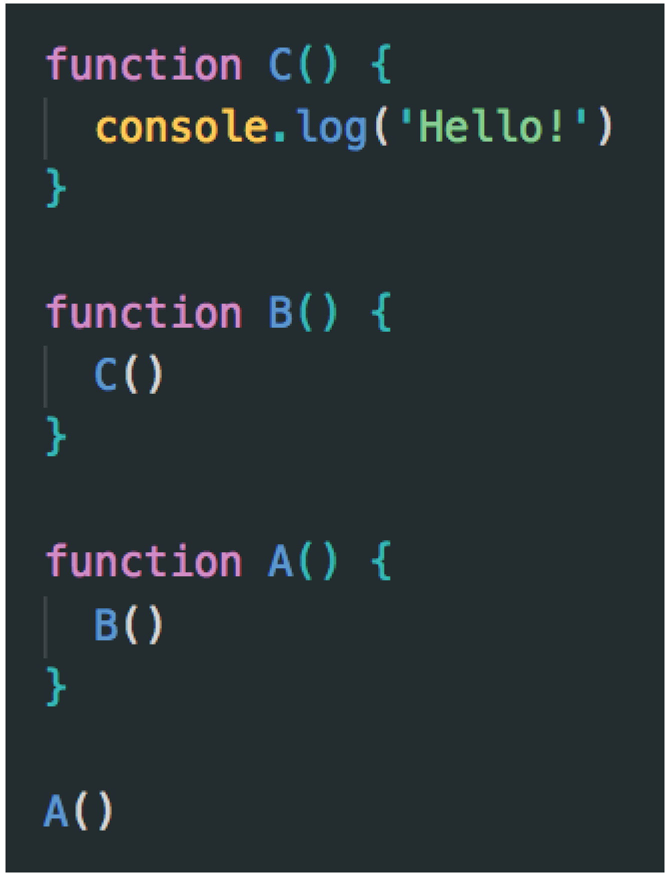
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



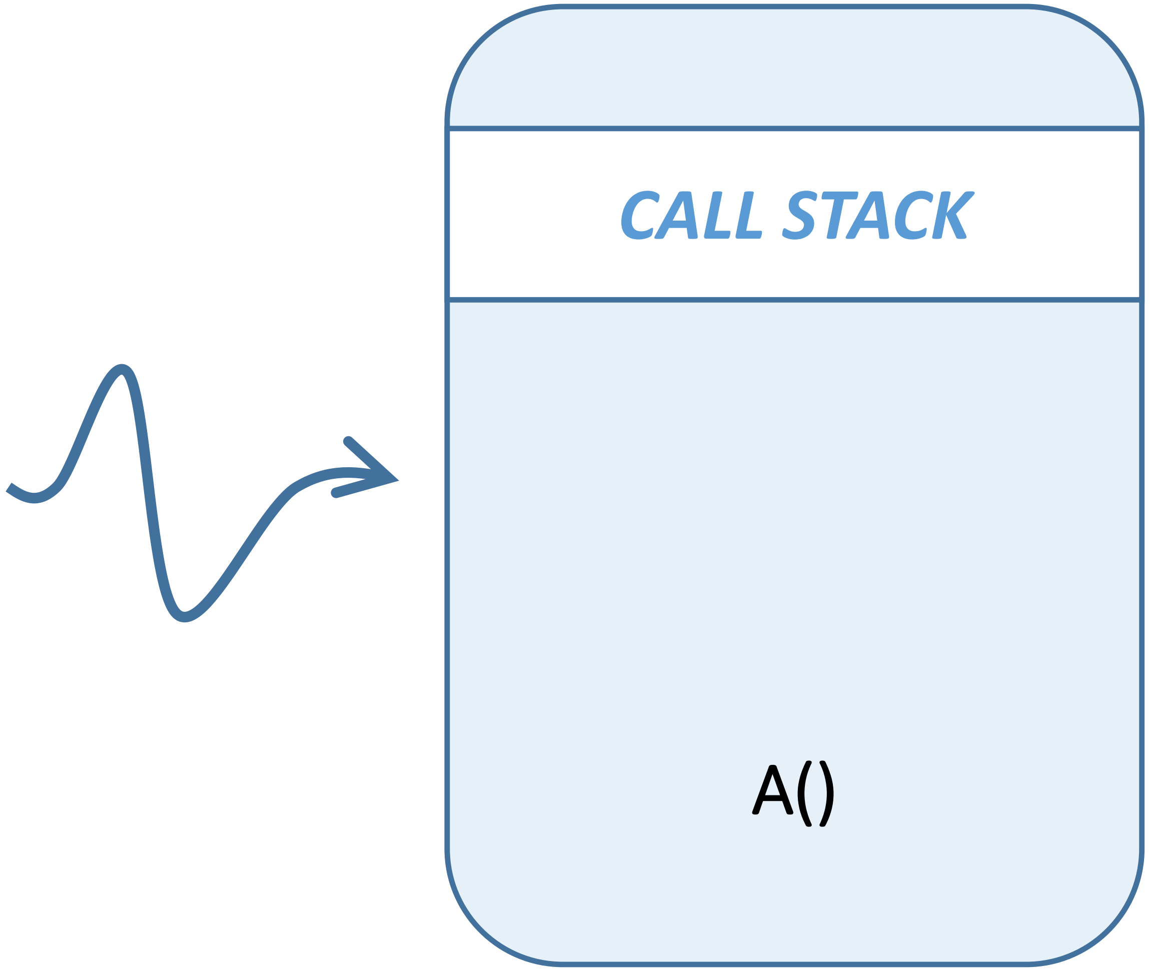


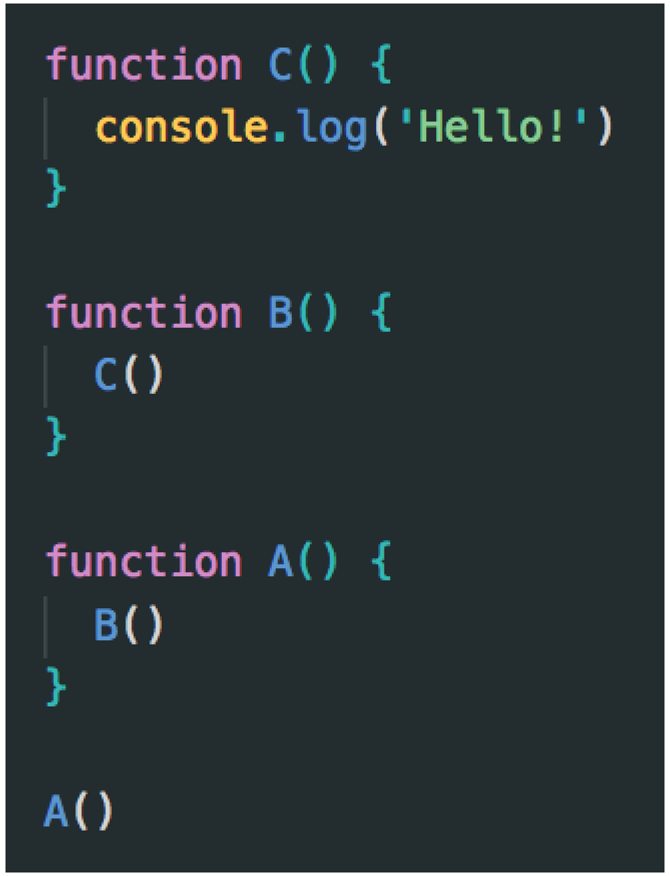
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



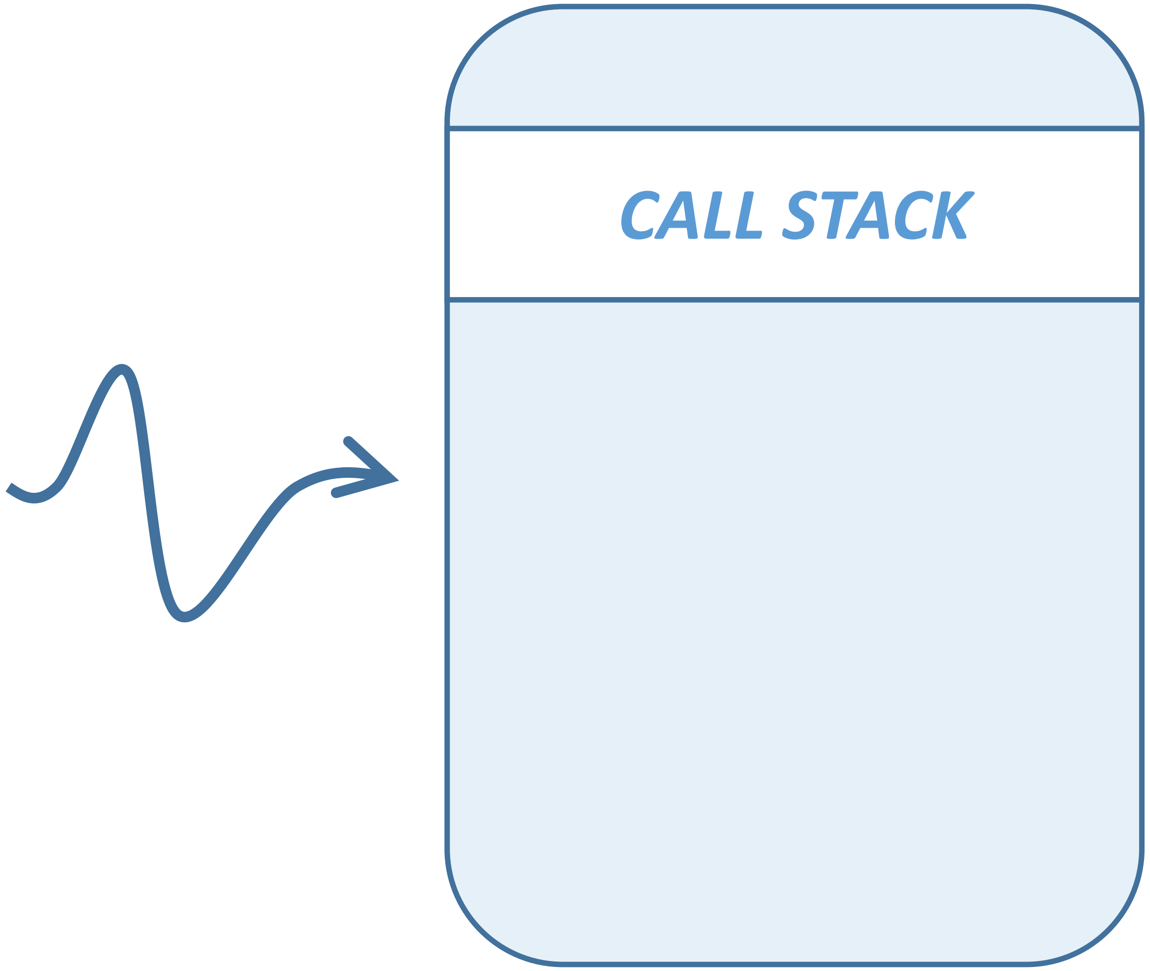


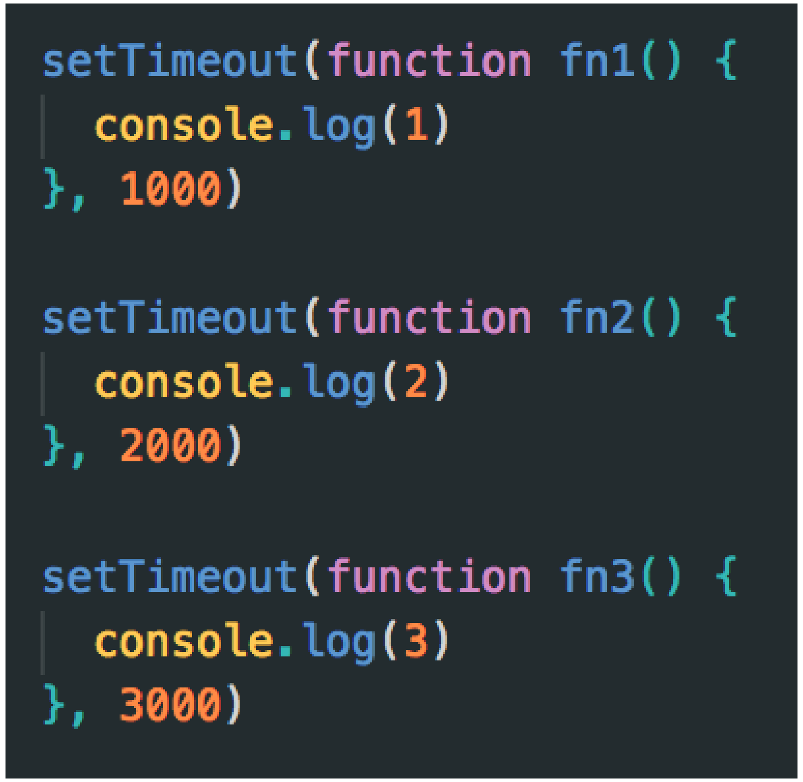
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```





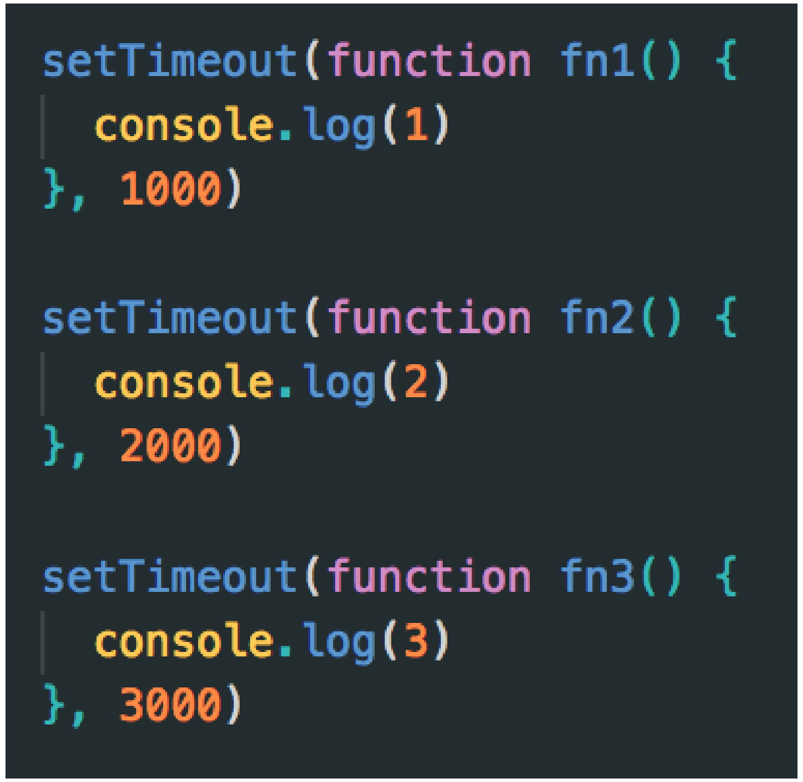
```
function C() {
  console.log('Hello!')
}

function B() {
  C()
}

function A() {
  B()
}

A()
```



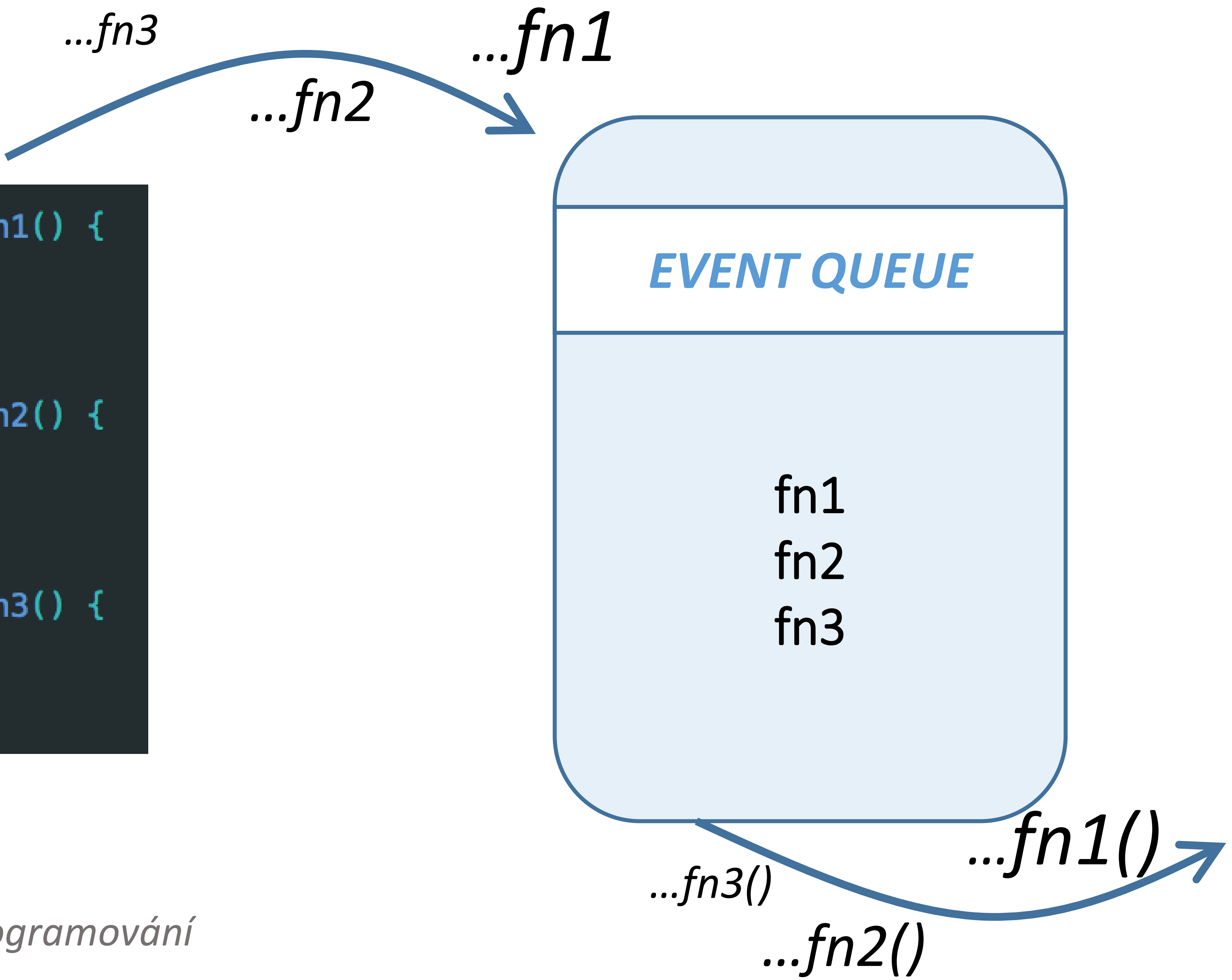


```
setTimeout(function fn1() {
  console.log(1)
}, 1000)

setTimeout(function fn2() {
  console.log(2)
}, 2000)

setTimeout(function fn3() {
  console.log(3)
}, 3000)
```

EVENT QUEUE



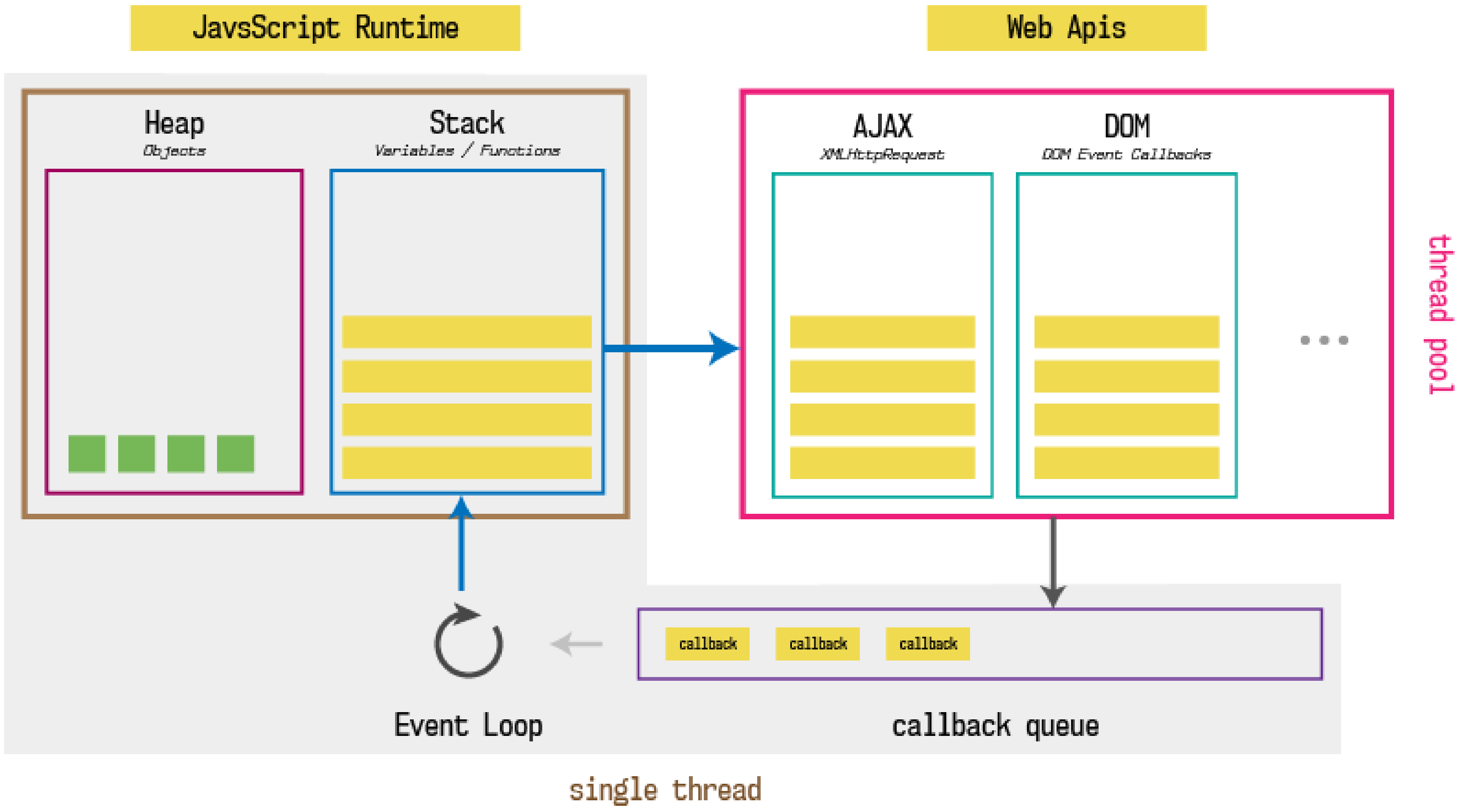
```
setTimeout(function fn1() {
  console.log(1)
}, 1000)

setTimeout(function fn2() {
  console.log(2)
}, 2000)

setTimeout(function fn3() {
  console.log(3)
}, 3000)
```



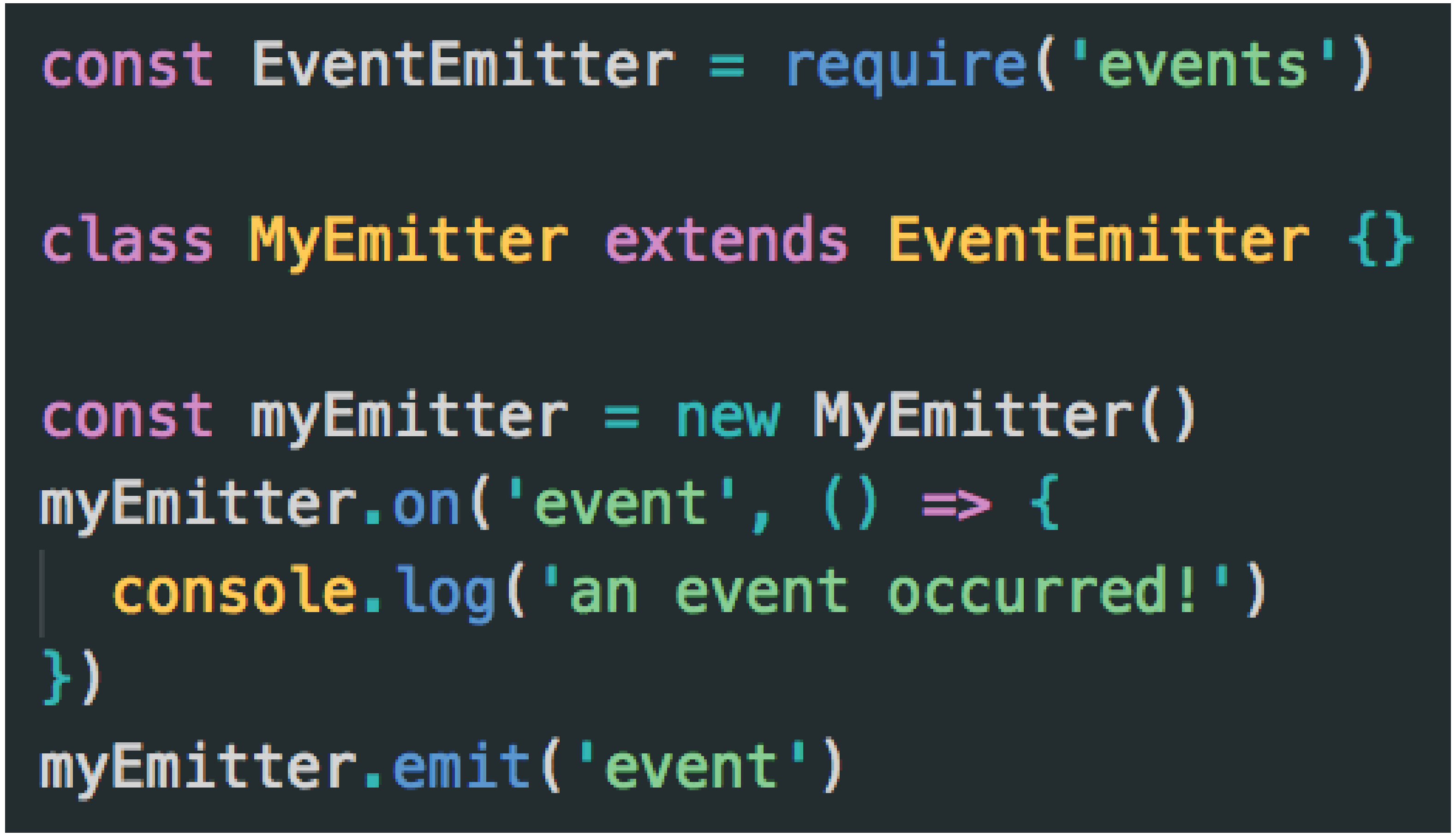
Fronta? Na co čekáme?

Fronta? Na co čekáme?

Na prázdný call-stack

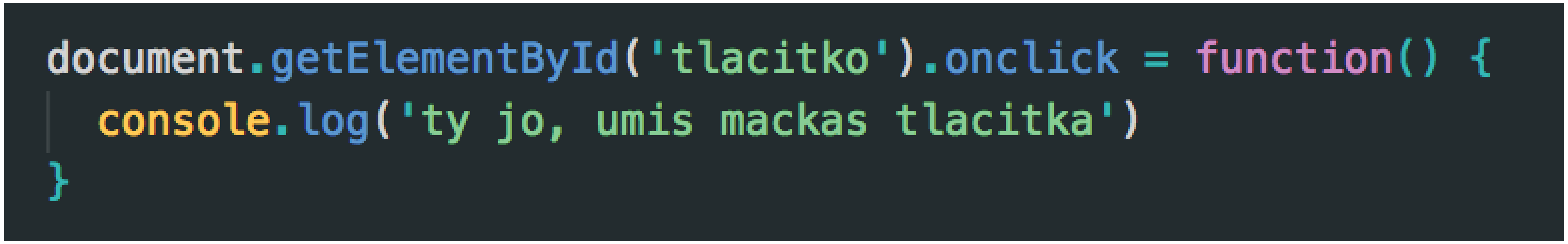

JavsScript Runtime
Web Apis
Heap
Objects
Stack
Variables / Functions
AJAX
XMLHttpRequest
DOM
DOM Event Callbacks
• • •
thread pool
callback
callback
callback
Event Loop
callback queue
single thread

EventEmitter, pub/sub

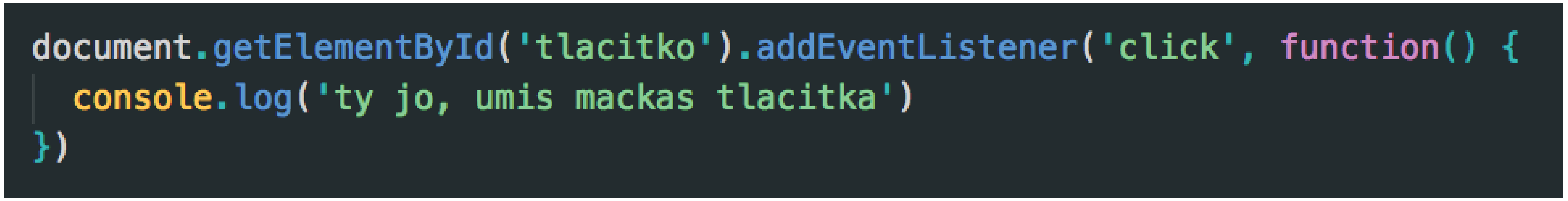


```
const EventEmitter = require('events')

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter()
myEmitter.on('event', () => {
  console.log('an event occurred!')
})
myEmitter.emit('event')
```

Registrace callbacku

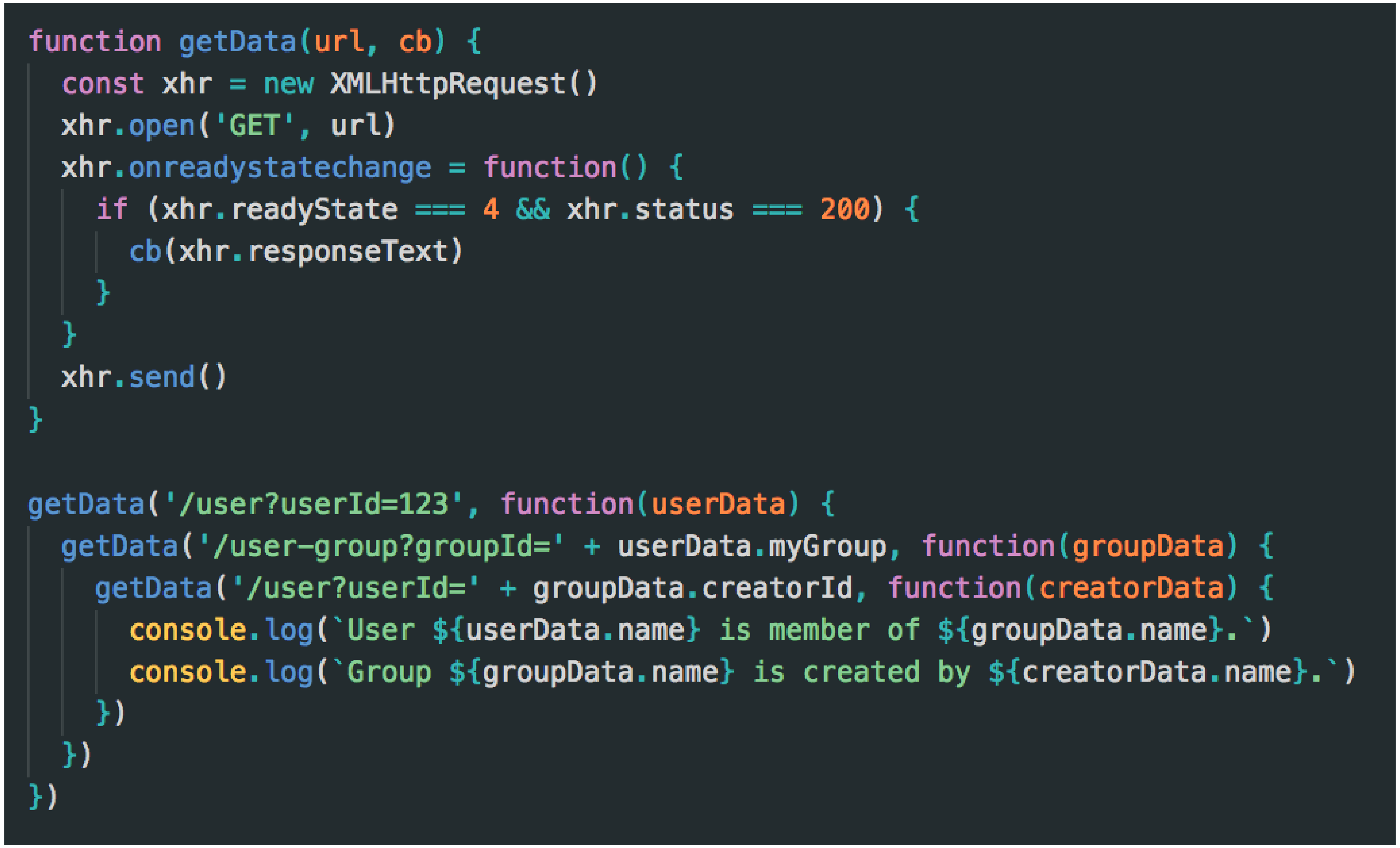


```
document.getElementById('tlacitko').onclick = function() {
  console.log('ty jo, umis mackas tlacitka')
}
```



```
document.getElementById('tlacitko').addEventListener('click', function() {
  console.log('ty jo, umis mackas tlacitka')
})
```

Callback hell



```
function getData(url, cb) {
  const xhr = new XMLHttpRequest()
  xhr.open('GET', url)
  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
      cb(xhr.responseText)
    }
  }
  xhr.send()
}

getData('/user?userId=123', function(userData) {
  getData('/user-group?groupId=' + userData.myGroup, function(groupData) {
    getData('/user?userId=' + groupData.creatorId, function(creatorData) {
      console.log(`User ${userData.name} is member of ${groupData.name}.`)
      console.log(`Group ${groupData.name} is created by ${creatorData.name}.`)
    })
  })
})
```

Promises, generatory a async/await

JS p.6 ⟶ *Asynchronní programování*

// end