你懂了 JavaScript，也不懂 JavaScript

Huli @ MOPCON 2021

About

OneDegree

上半年：OneDegree 前端工程師

About

OneDegree

上半年：OneDegree 前端工程師

Cymetrics

下半年：Cymetrics 資安工程師

開始之前，先來個小測驗

下列各組「」內的字，讀音相同的選項是

- (A) 令人發「噱」 / 「遽」然而逝
- (B) 同心「戮」力 / 「蓼」菜成行
- (C) 寒「蛩」鳴秋 / 「煢」居荒野
- (D)「瓠」巴鼓瑟 / 簞「瓢」屢空

出處：101 年指考國文

下列各組「」內的字，讀音相同的選項是

- (A) 令人發「噱」 / 「遽」然而逝
- (B) 同心「戮」力 / 「蓼」菜成行
- (C) 寒「蛩」鳴秋 / 「煢」居荒野
- (D)「瓠」巴鼓瑟 / 簞「瓢」屢空

出處：101 年指考國文

下列各組 相同的選項是

- (A) 而逝
- (B) 戍行
- (C) 荒野
- (D) 憂空

出處：101 年指考國文

下列各組 的選項是

-
-
-
-

出處：101 年指考國文

讓我們再來一個小測驗

請問以下四個的結果是

```
[] + []
```

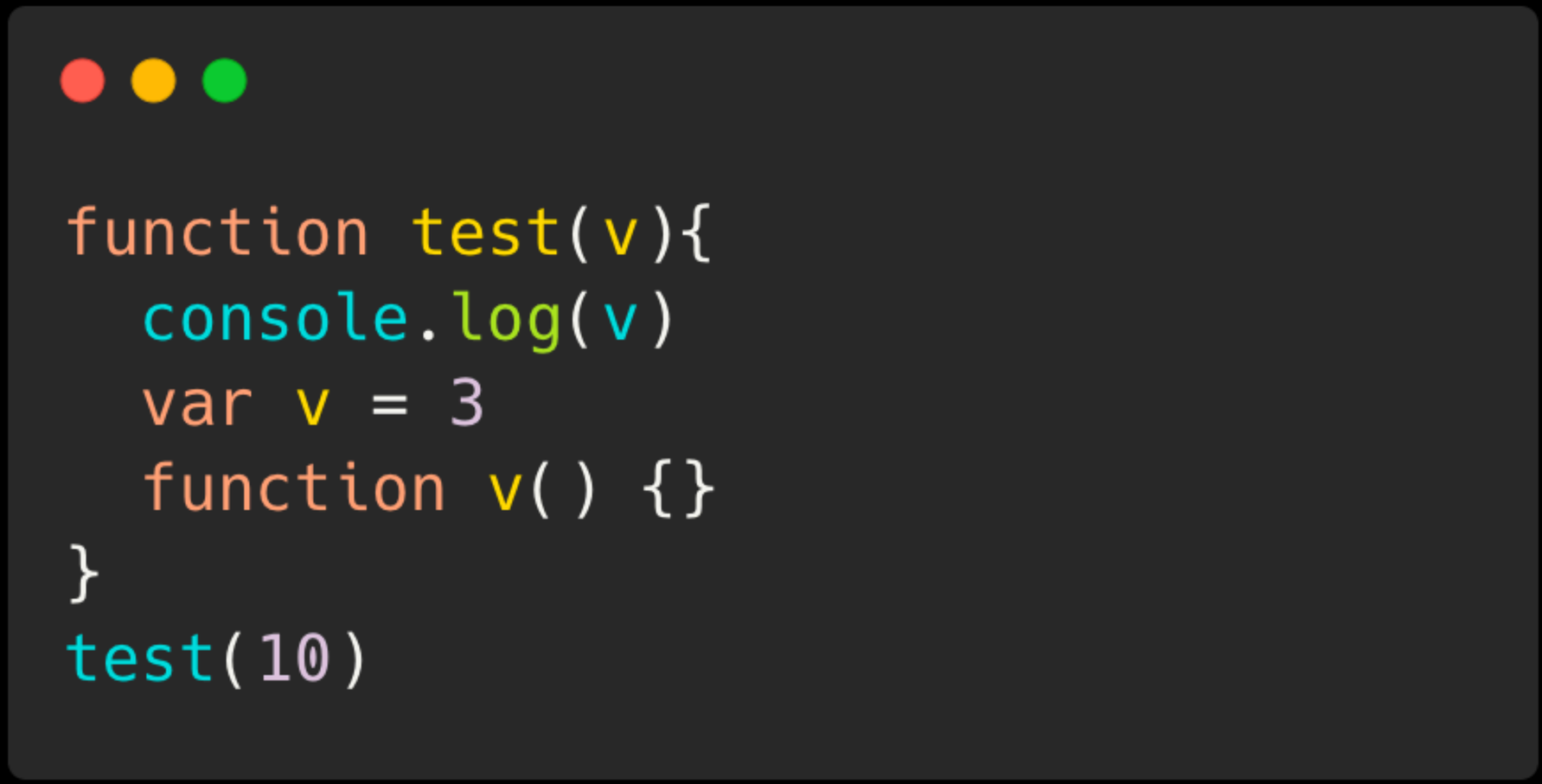
```
[] + {}
```

```
{} + []
```

```
{} + {}
```

請問以下四個的結果是

1. 這個對我寫 code 有幫助嗎？
2. 不知道的話，會容易寫出 bug 嗎？



```
function test(v){
  console.log(v)
  var v = 3
  function v() {}
}
test(10)
```


var a = b = 3

eslint: b is not defined


function getPosts() {
fetch(API_URL)
.then(res => res.json())
.then(data => {
return data
})
}
const data = getPosts()

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
```

```
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```

1. 比較不重要但有趣的知識
2. 重要的知識

成為 JS 大師 aka spec 大師

1. 比較不重要但有趣的知識
2. 重要的知識

1. 比較不重要但有趣的知識
2. 重要的知識

避免寫出 bug，寫出更好的 code

時間有限的前提下，挑價值最高的先學

Agenda

1. Type
2. Hoisting
3. This
4. Prototype
5. Async

Type

Number 跟 String

```
[1,5,3,11,7].sort()
```

```
[1,5,3,11,7].sort()
[1,11,3,5,7]
```

金額：

金額：

金額：

總金額：

確定

```
btn.onclick = () => {
  let total = 0
  const inputs =
    document.querySelectorAll('.n')
  const expected =
    document.querySelector('.t').value
  for(let input of inputs) {
    total += input.value
  }
  alert(total === expected)
}
```

```
btn.onclick = () => {
  let total = 0
  const inputs =
    document.querySelectorAll('.n')
  const expected =
    Number(document.querySelector('.t').value)
  for(let input of inputs) {
    total += Number(input.value)
  }
  alert(total === expected)
}
```

金額：1.1

金額：1.1

金額：1.1

總金額：3.3

確定

3.3000000000000003

Number.EPSILON = 2^-52

```
Math.abs(0.1+0.2-0.3) < Number.EPSILON
```

Turkxyy: EZ

Doujins: KKona BROTHER

Akuretaki: EZ WICKED

controlla408: HARD CARRY pog

DudeNamedTree: Got carried, on gawd

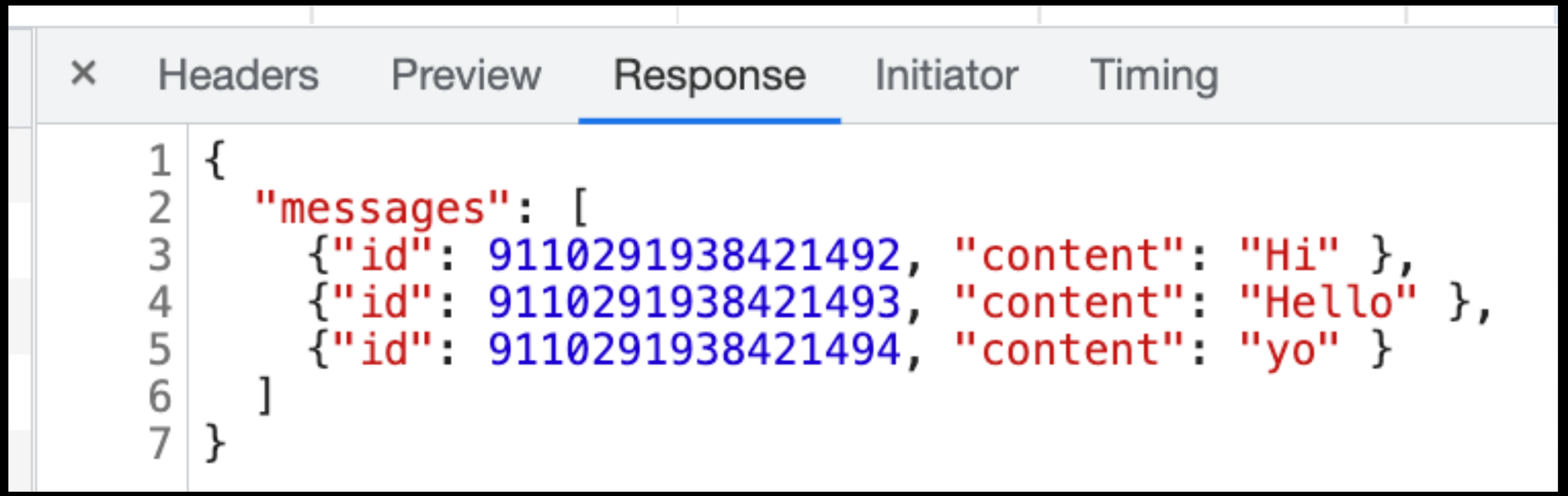
KhristophesaurusDinosarus: BELIEVERS EZ EZ BELIEVERS EZ EZ BELIEVERS EZ EZ BELIEVERS EZ EZ BELIEVERS EZ EZ BELIEVERS EZ EZ BELIEVERS EZ EZ

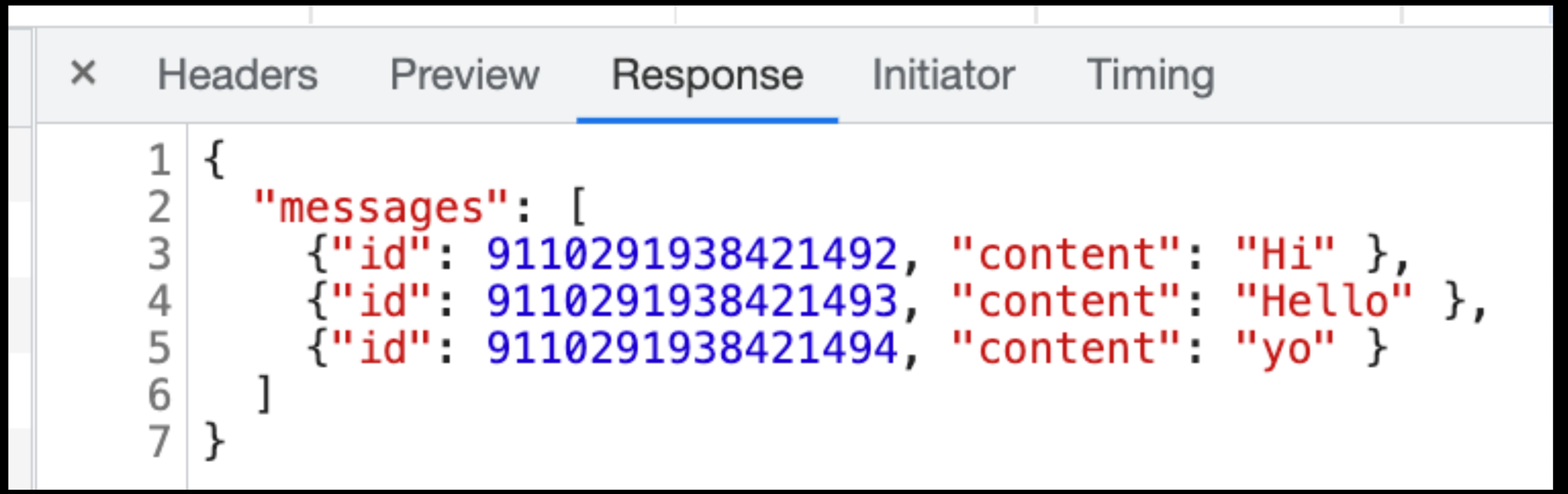
Akuretaki: EZ WICKED EZ WICKED

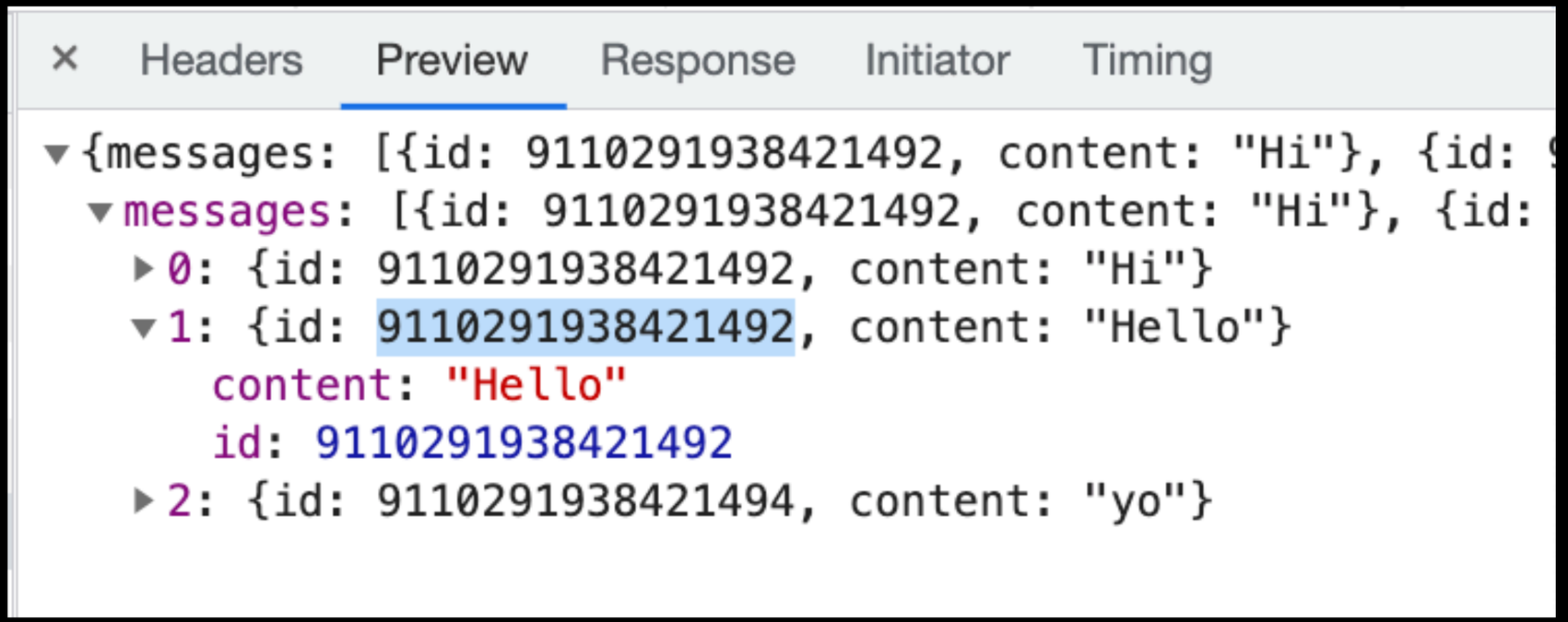
Tu4k_: SHEESHHHh + 13k EZ

nobodyhearts: that was ridiculously


{
"messages": [
{"id": 9110291938421492, "content": "Hi" },
{"id": 9110291938421493, "content": "Hello" },
{"id": 9110291938421494, "content": "yo" }
]
}


Headers
Preview
Response
Initiator
Timing
{
"messages": [
{"id": 9110291938421492, "content": "Hi" },
{"id": 9110291938421493, "content": "Hello" },
{"id": 9110291938421494, "content": "yo" }
]
}


Headers
Preview
Response
Initiator
Timing
{
"messages": [
{"id": 9110291938421492, "content": "Hi" },
{"id": 9110291938421493, "content": "Hello" },
{"id": 9110291938421494, "content": "yo" }
]
}


Headers
Preview
Response
Initiator
Timing
{messages: [{id: 9110291938421492, content: "Hi"}, {id:
messages: [{id: 9110291938421492, content: "Hi"}, {id:
0: {id: 9110291938421492, content: "Hi"}
1: {id: 9110291938421492, content: "Hello"}
content: "Hello"
id: 9110291938421492
2: {id: 9110291938421494, content: "yo"}

Number.MAX_SAFE_INTEGER

9007199254740991

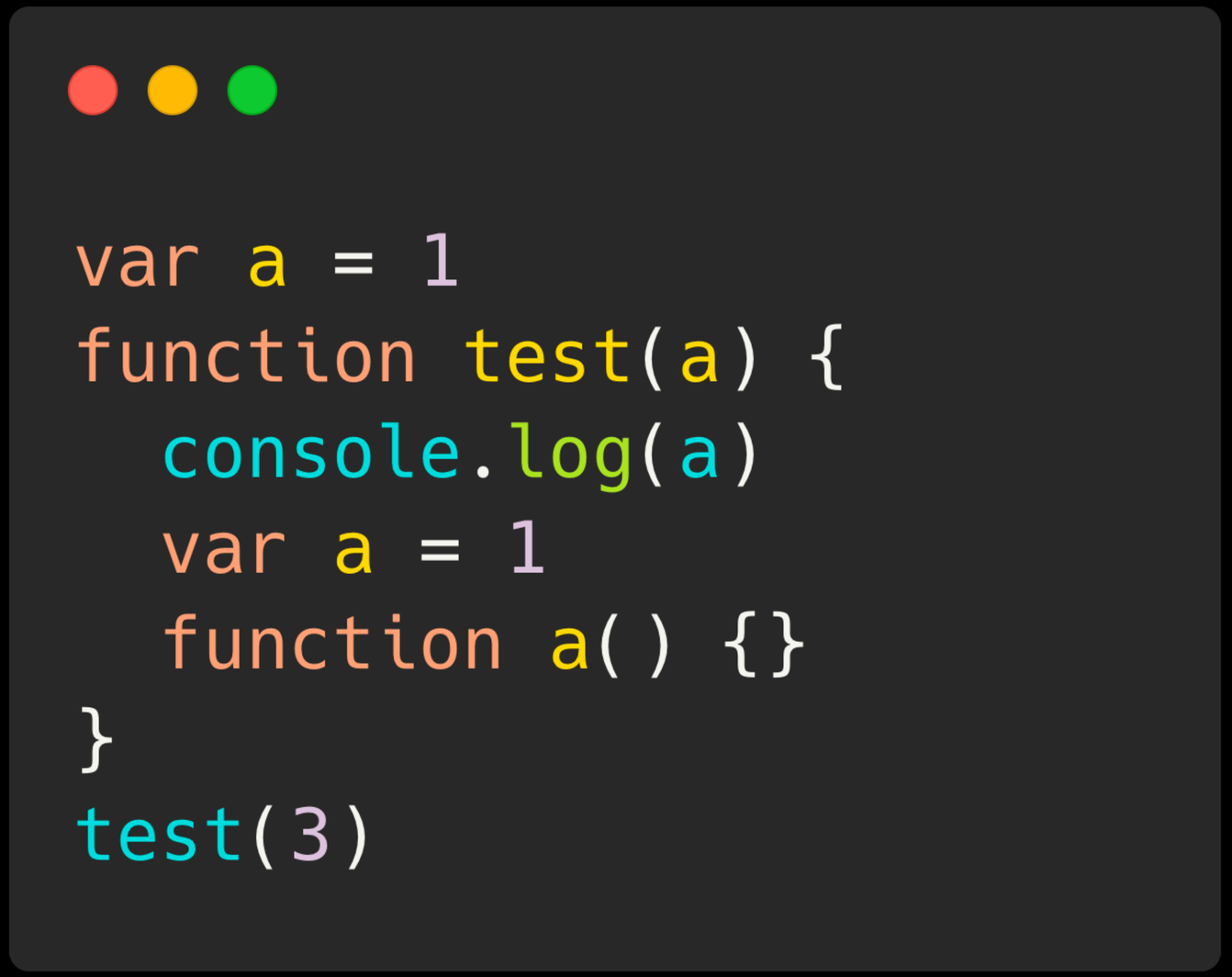
第七種資料型別

BigInt

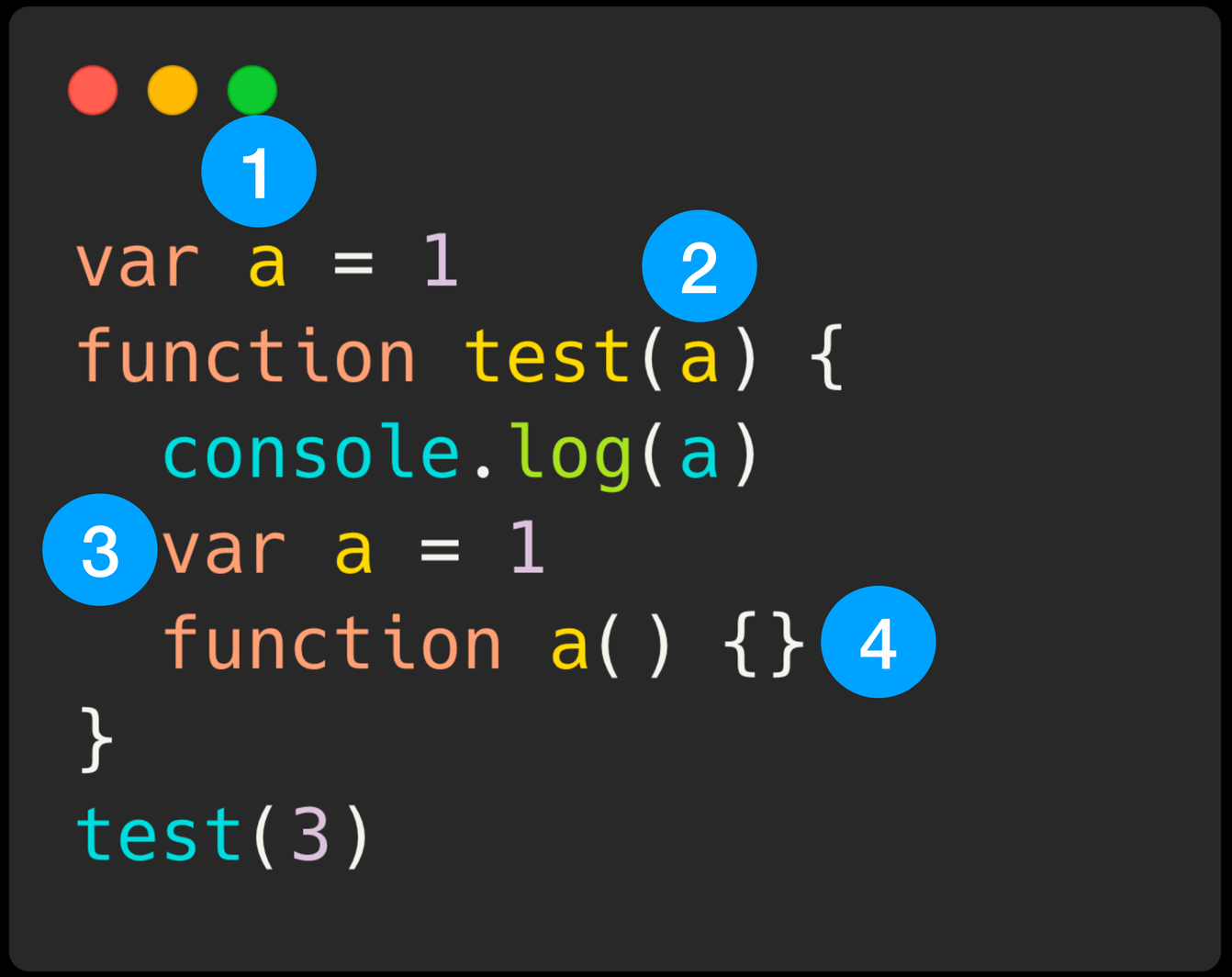
重點

1. 排序時注意比較的是字典序
2. 注意數字加字串會變字串
3. 處理小數時注意浮點數問題
4. 處理大數時注意數字範圍
5. 大數可以用 BigInt

Hoisting

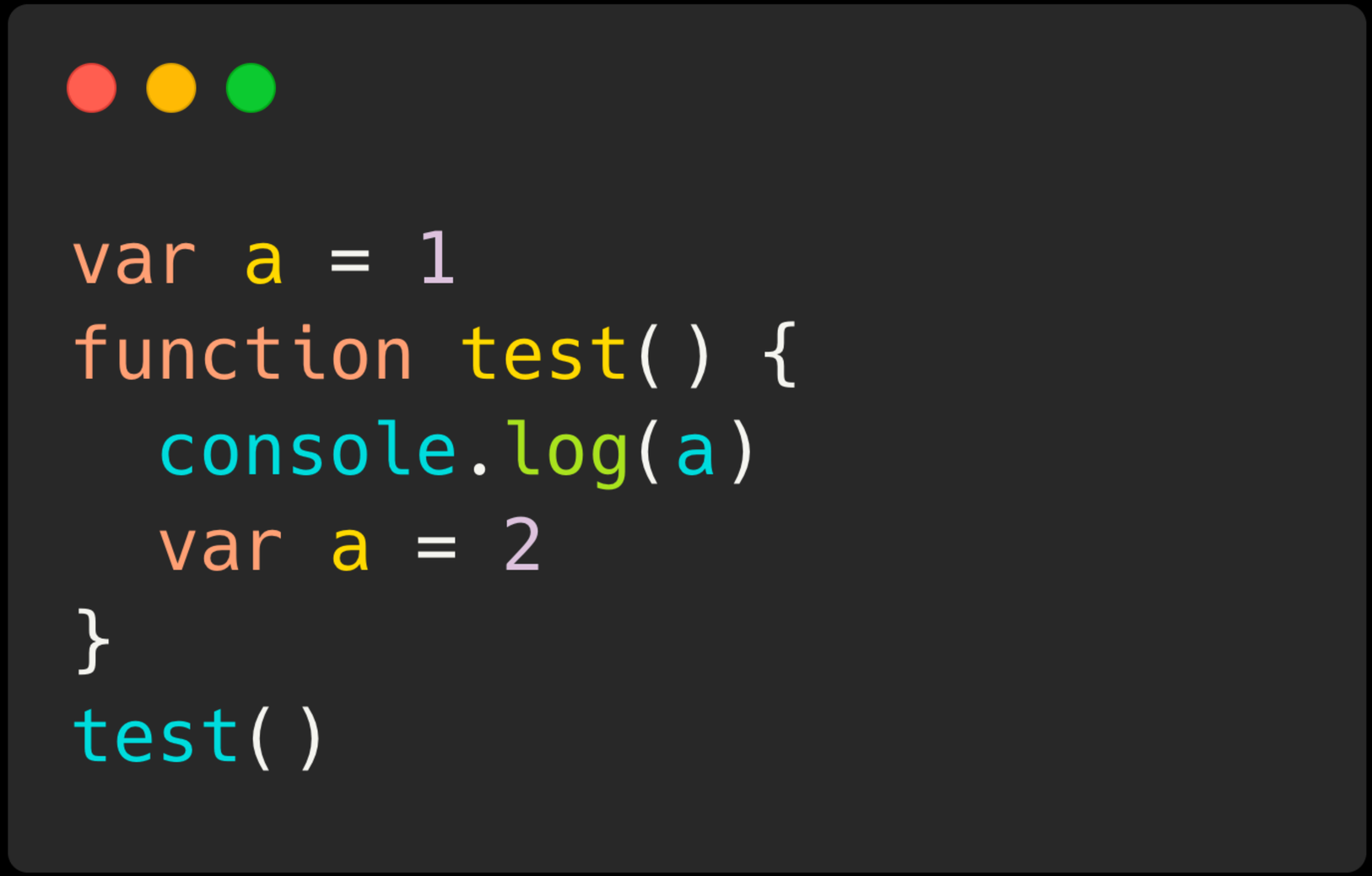


```
var a = 1
function test(a) {
  console.log(a)
  var a = 1
  function a() {}
}
test(3)
```


1
var a = 1
2
function test(a) {
console.log(a)
3
var a = 1
function a() {}
4
}
test(3)

```
var a
funct
  con
var
  fun
}
test(
```

JS 在執行前就會做一些處理

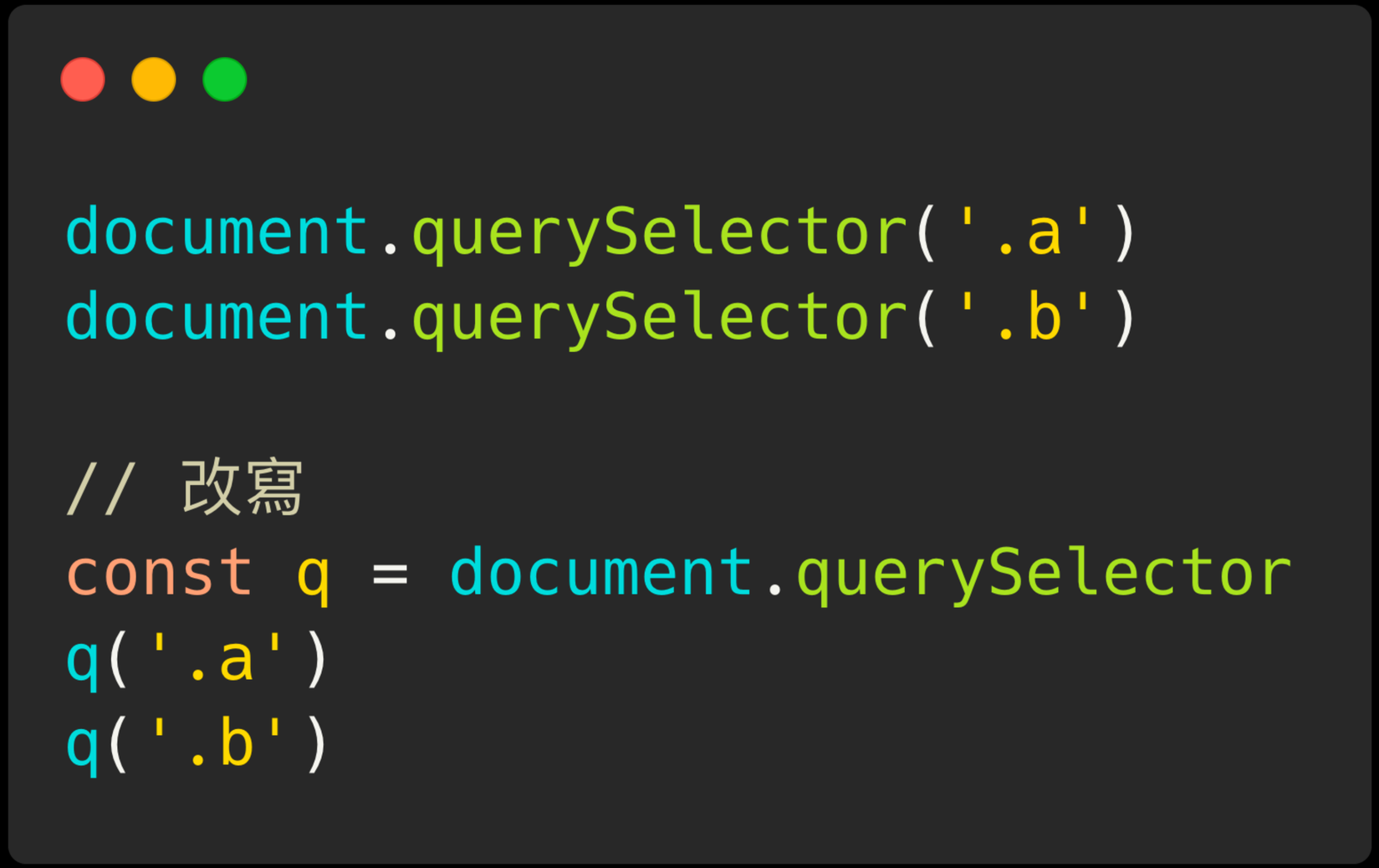


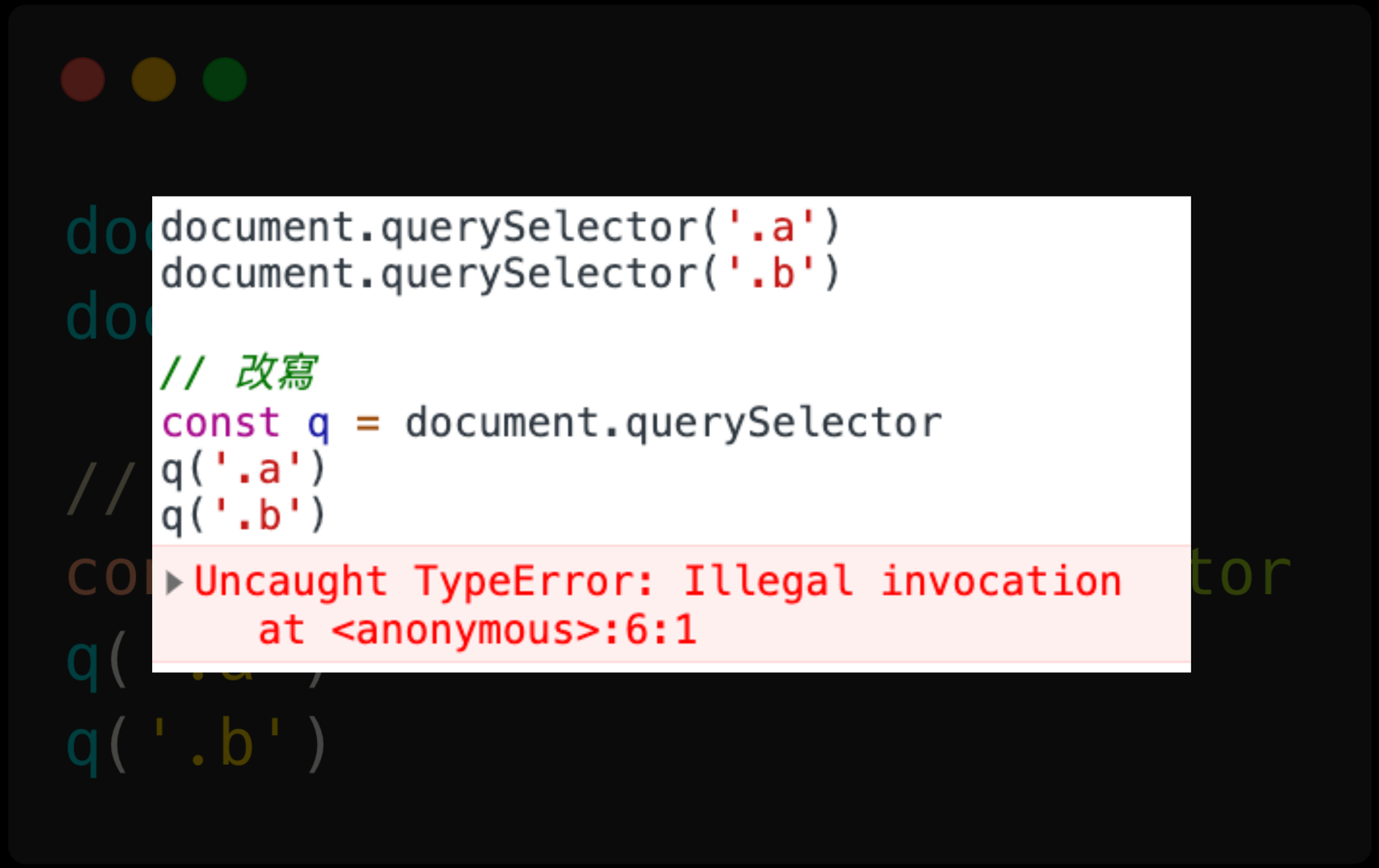
```
var a = 1
function test() {
  console.log(a)
  var a = 2
}
test()
```

重點

1. JS 還是有編譯的存在
2. 在進入函式時就會對宣告做處理

This


document.querySelector('.a')
document.querySelector('.b')
// 改寫
const q = document.querySelector
q('.a')
q('.b')


document.querySelector('.a')
document.querySelector('.b')
// 改寫
const q = document.querySelector
q('.a')
q('.b')
Uncaught TypeError: Illegal invocation
at <anonymous>:6:1

```
document.query()

var q = document.query
q()
```

```
document.query()

var q = document.query
q()
```

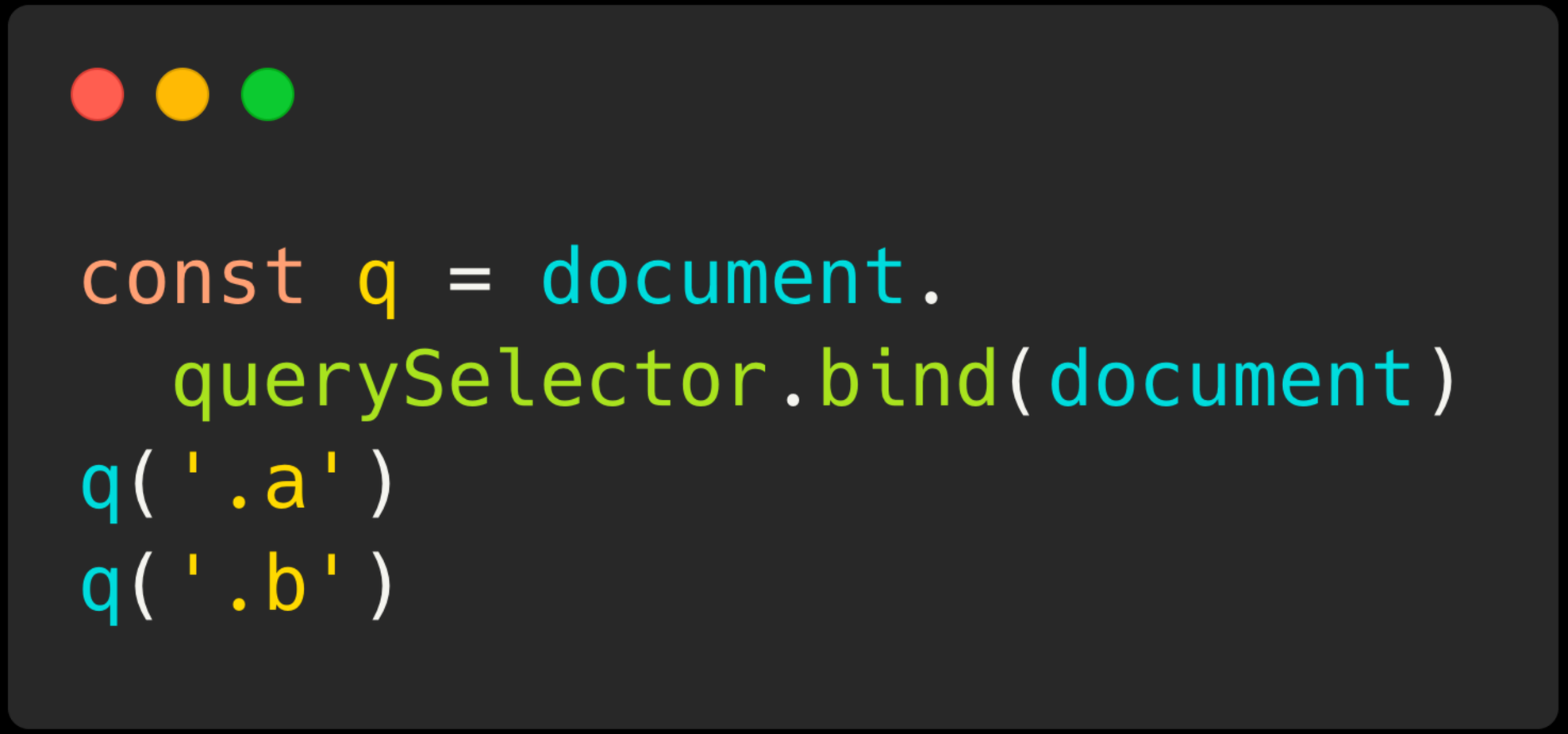
this 的值跟怎麼呼叫函式有關

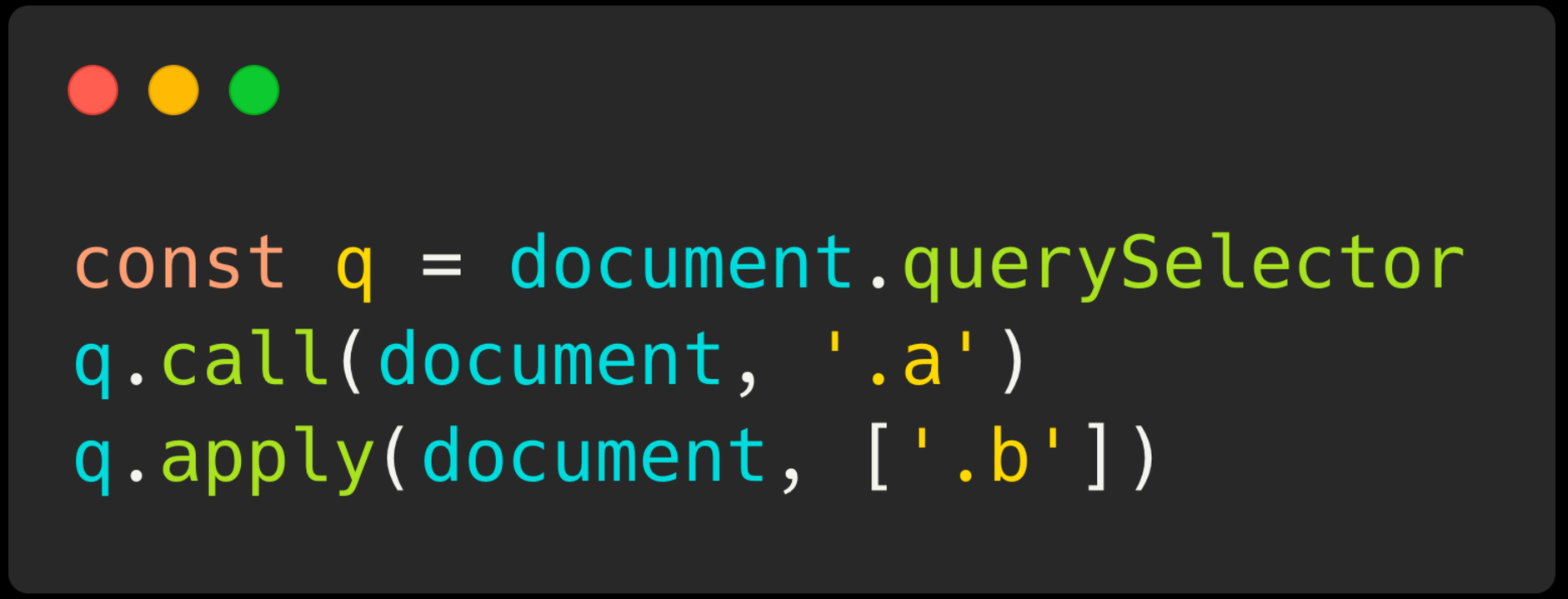

```
document.query()

var q = document.query
q()
```

this

```
document.query()

var q = document.query
q()
```

this?


const q = document.
querySelector.bind(document)
q('.a')
q('.b')

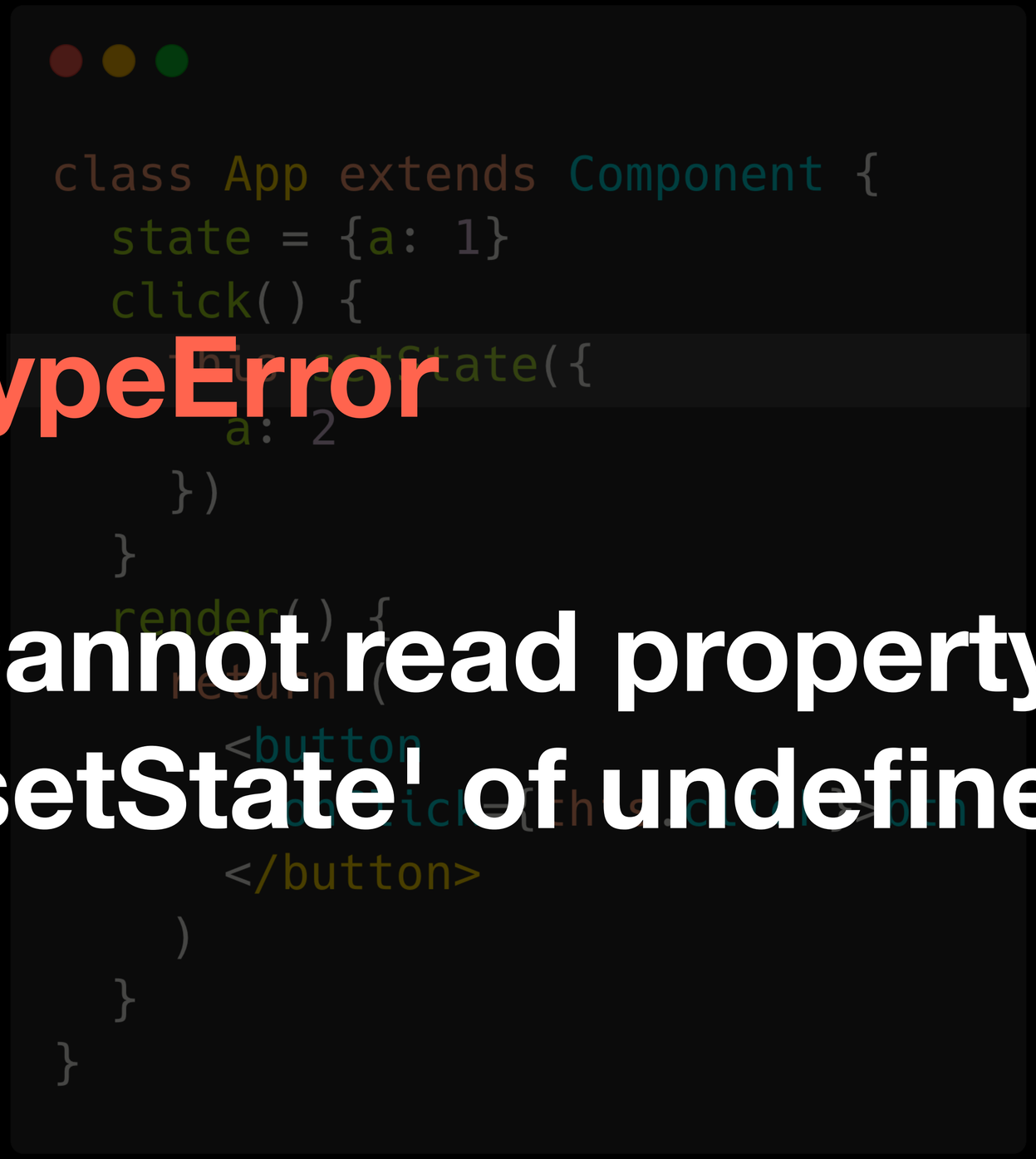

const q = document.querySelector
q.call(document, '.a')
q.apply(document, ['.b'])

```
class App extends Component {
  state = {a: 1}
  click() {
    this.setState({
      a: 2
    })
  }
  render() {
    return (
      <button
        onClick={this.click}>btn
      </button>
    )
  }
}
```

```
class App extends Component {
  state = {a: 1}
  click() {
    this.setState({
      a: 2
    })
  }
  render() {
    return (
      <button
        onClick={this.click}>btn
      </button>
    )
  }
}
```

```
class App extends Component {
  state = {a: 1}
  click() {
    this.setState({
      a: 2
    })
  }
  render() {
    return (
      <button
        onClick={this.click}>btn
      </button>
    )
  }
}
```

```
class App extends Component {
  state = {a: 1}
  click() {
    this.setState({
      a: 2
    })
  }
  render() {
    return (
      <button
        onClick={this.click}>btn
      </button>
    )
  }
}
```

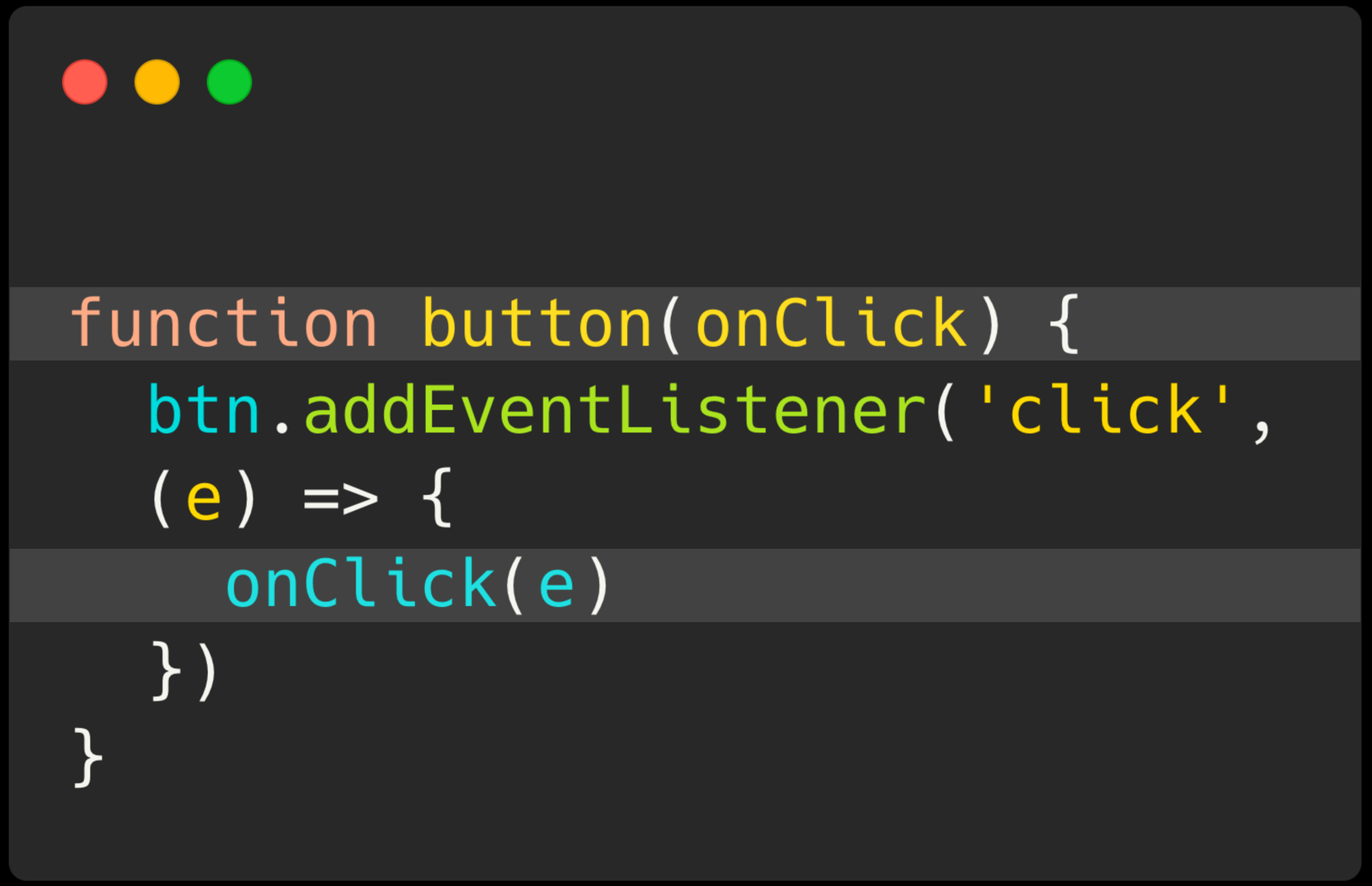


TypeError

Cannot read property 'setState' of undefined

this 的值跟怎麼呼叫函式有關

```
class App extends Component {
  state = {a: 1}
  click() {
    this.setState({
      a: 2
    })
  }
  render() {
    return (
      <button
        onClick={this.click}>btn
      </button>
    )
  }
}
```



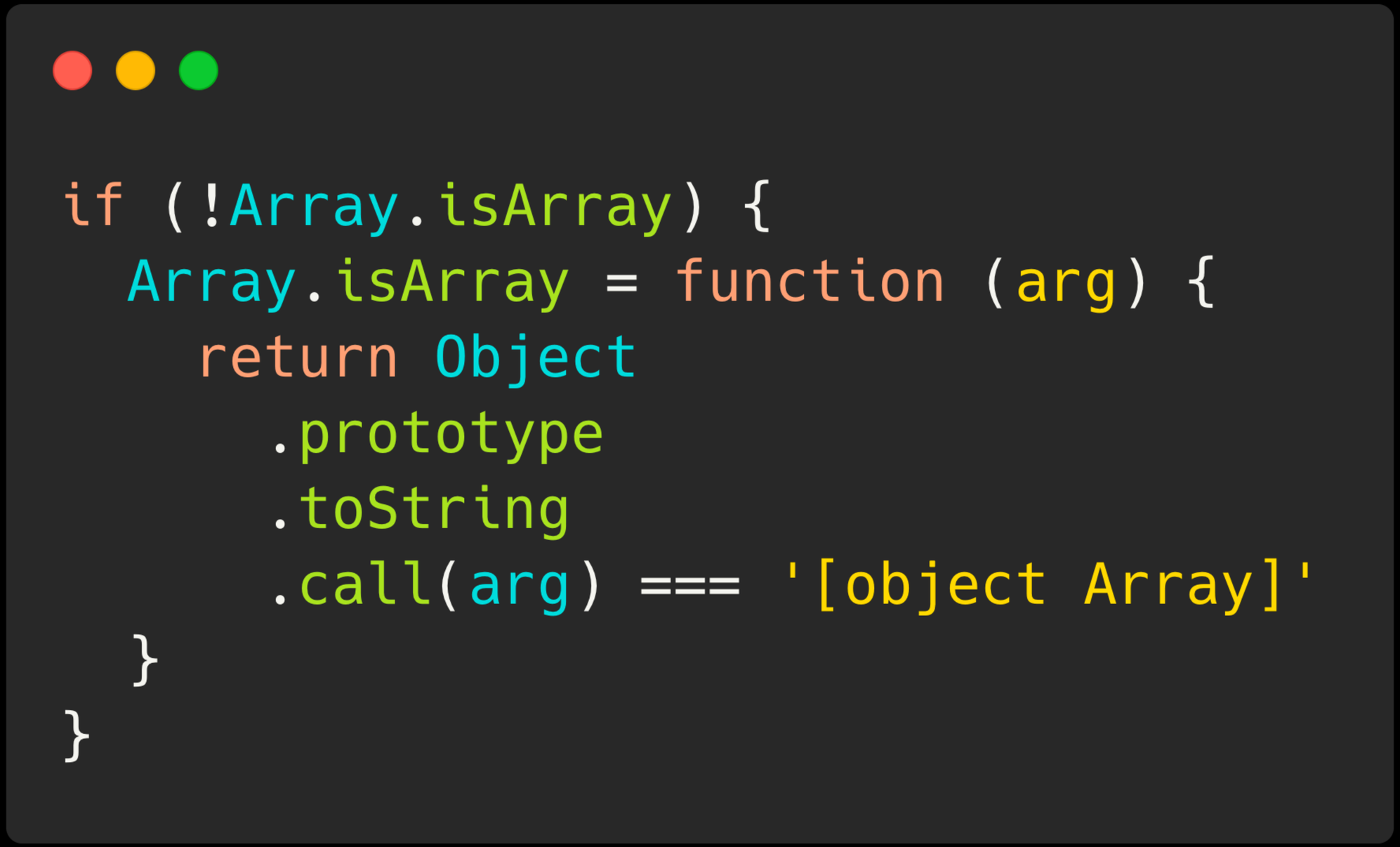
```
function button(onClick) {
  btn.addEventListener('click',
  (e) => {
    onClick(e)
  })
}
```

```
class App extends Component {
  state = {a: 1}
  click() {
    this.setState({
      a: 2
    })
  }
  render() {
    return (
      <button
        onClick={() => this.click()}>btn
      </button>
    )
  }
}
```

重點

1. 呼叫函式的方式會影響 this
2. bind 可以綁定特定 this
3. call、apply 呼叫函式時可以指定 this
4. arrow function 也會改變 this

Prototype



```
if (!Array.isArray) {
  Array.isArray = function (arg) {
    return Object
      .prototype
      .toString
      .call(arg) === '[object Array]'
  }
}
```

```
(1).toString()
```

```
(1).toString()

(1).__proto__
=> Number.prototype
```

```
(1).toString()
```

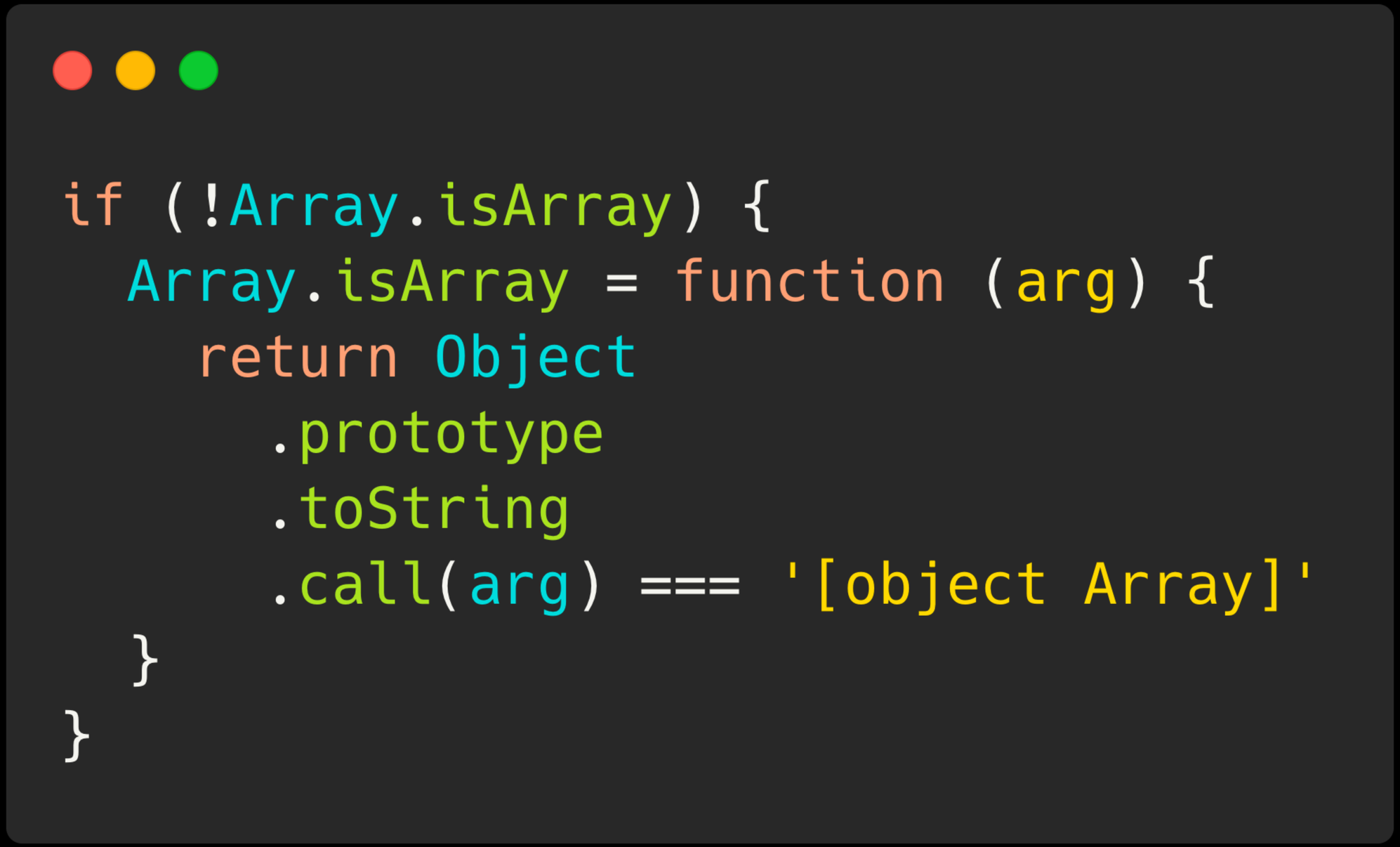
```
(1).__proto__
=> Number.prototype
```

```
Number.prototype.toString
```

```
(1).toString()

(1).__proto__
=> Number.prototype

Number.prototype.toString
```

Array.prototype.fill()

`fill()` 方法會將陣列中索引的第一個到最後一個的每個位置全部填入一個靜態的值。



```
if (!Array.isArray) {
  Array.isArray = function (arg) {
    return Object
      .prototype
      .toString
      .call(arg) === '[object Array]'
  }
}
```

```
function test(){
  var arr = Array.prototype.slice.apply(arguments)
  console.log(arr.join()) // 1, 2, 3
}
test(1, 2, 3)
```

23.1.3.25 Array.prototype.slice (*start*, *end*)

The `slice` method returns an array containing the elements of the array from element *start* up to, but not including, element *end* (or through the end of the array if *end* is **undefined**). If *start* is negative, it is treated as *length* + *start* where *length* is the length of the array. If *end* is negative, it is treated as *length* + *end* where *length* is the length of the array.

When the `slice` method is called, the following steps are taken:

1. Let *O* be ? ToObject(**this** value).
2. Let *len* be ? LengthOfArrayLike(*O*).

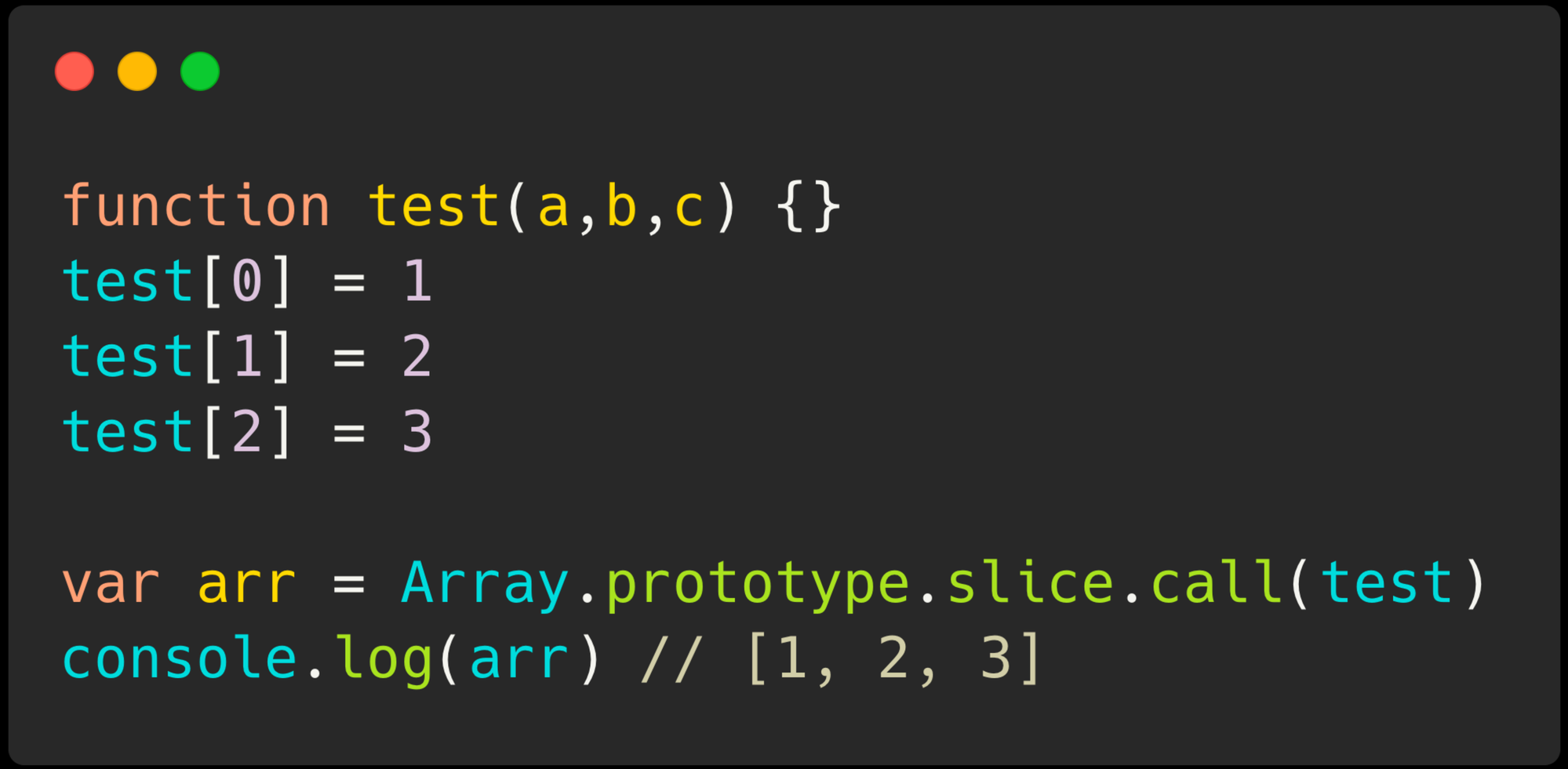
NOTE 2 The `slice` function is intentionally generic; it does not require that its **this** value be an Array. Therefore it can be transferred to other kinds of objects for use as a method.

7.3.18 LengthOfArrayLike (*obj*)

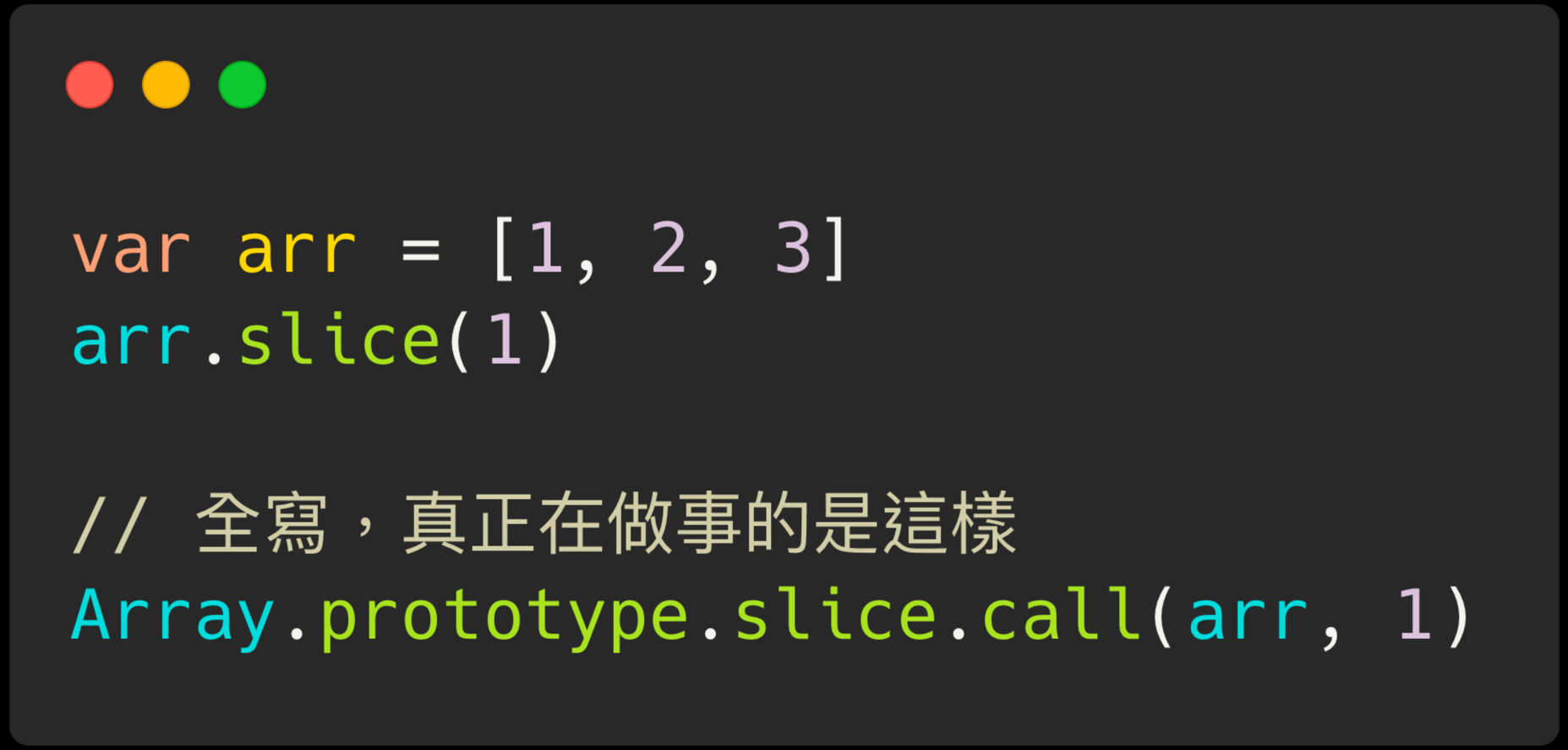
The abstract operation LengthOfArrayLike takes argument *obj* (an Object). It returns the value of the **"length"** property of an array-like object (as a non-negative integer). It performs the following steps when called:

1. Return ℝ(? ToLength(? Get(*obj*, **"length"**))).

An *array-like object* is any object for which this operation returns an integer rather than an abrupt completion.



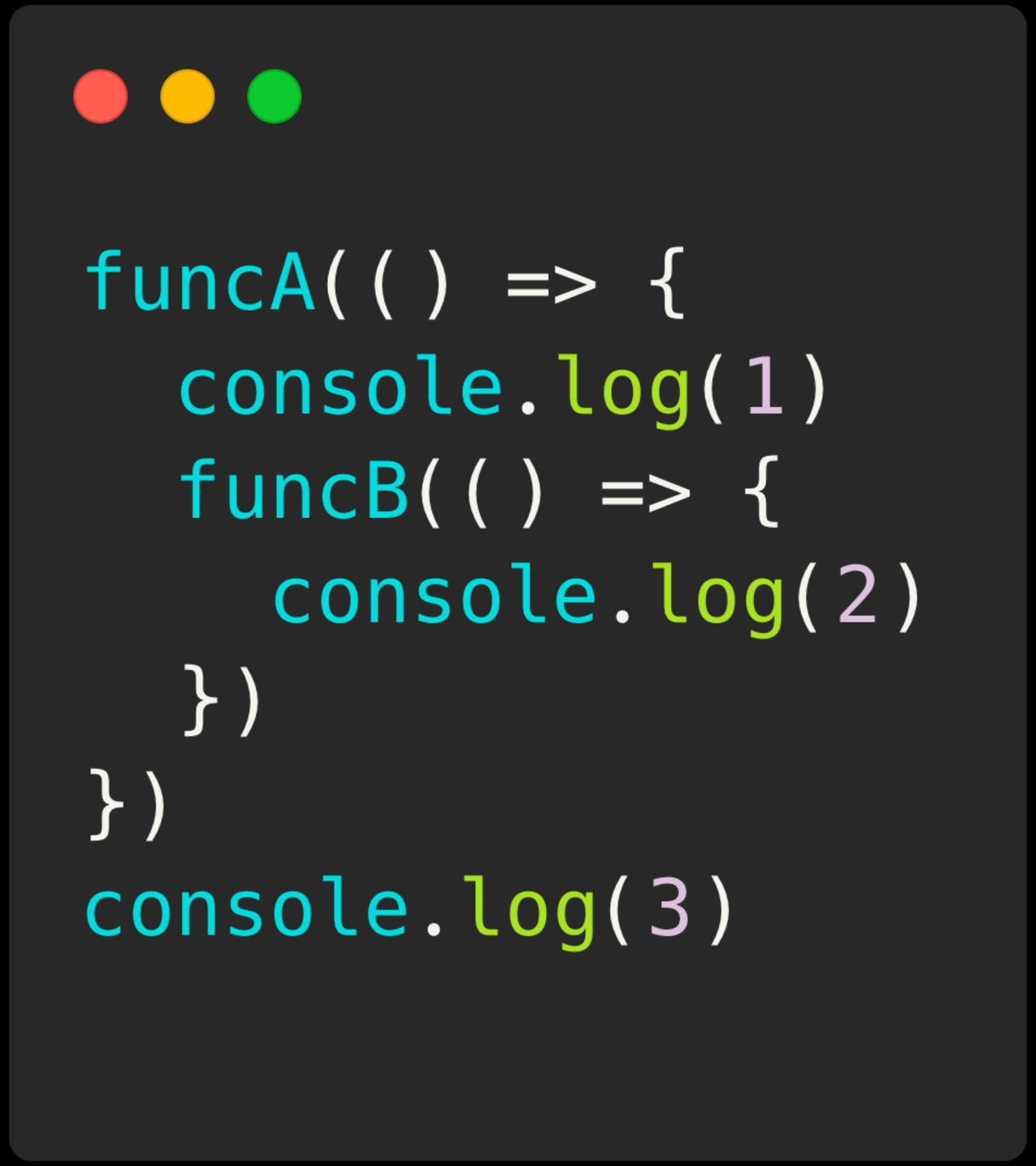
```
function test(a,b,c) {}
test[0] = 1
test[1] = 2
test[2] = 3

var arr = Array.prototype.slice.call(test)
console.log(arr) // [1, 2, 3]
```

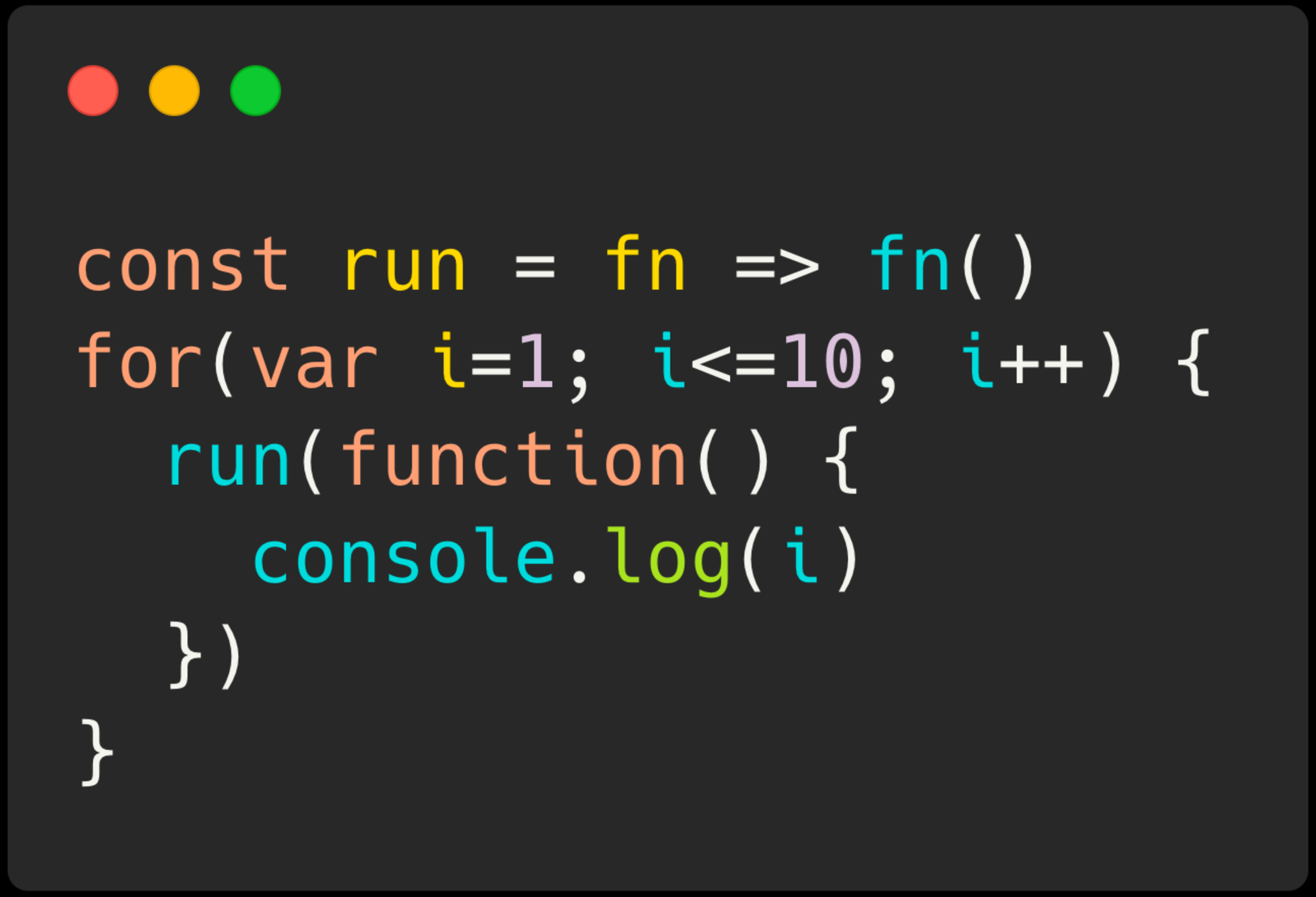

var arr = [1, 2, 3]
arr.slice(1)
// 全寫，真正在做事的是這樣
Array.prototype.slice.call(arr, 1)

重點

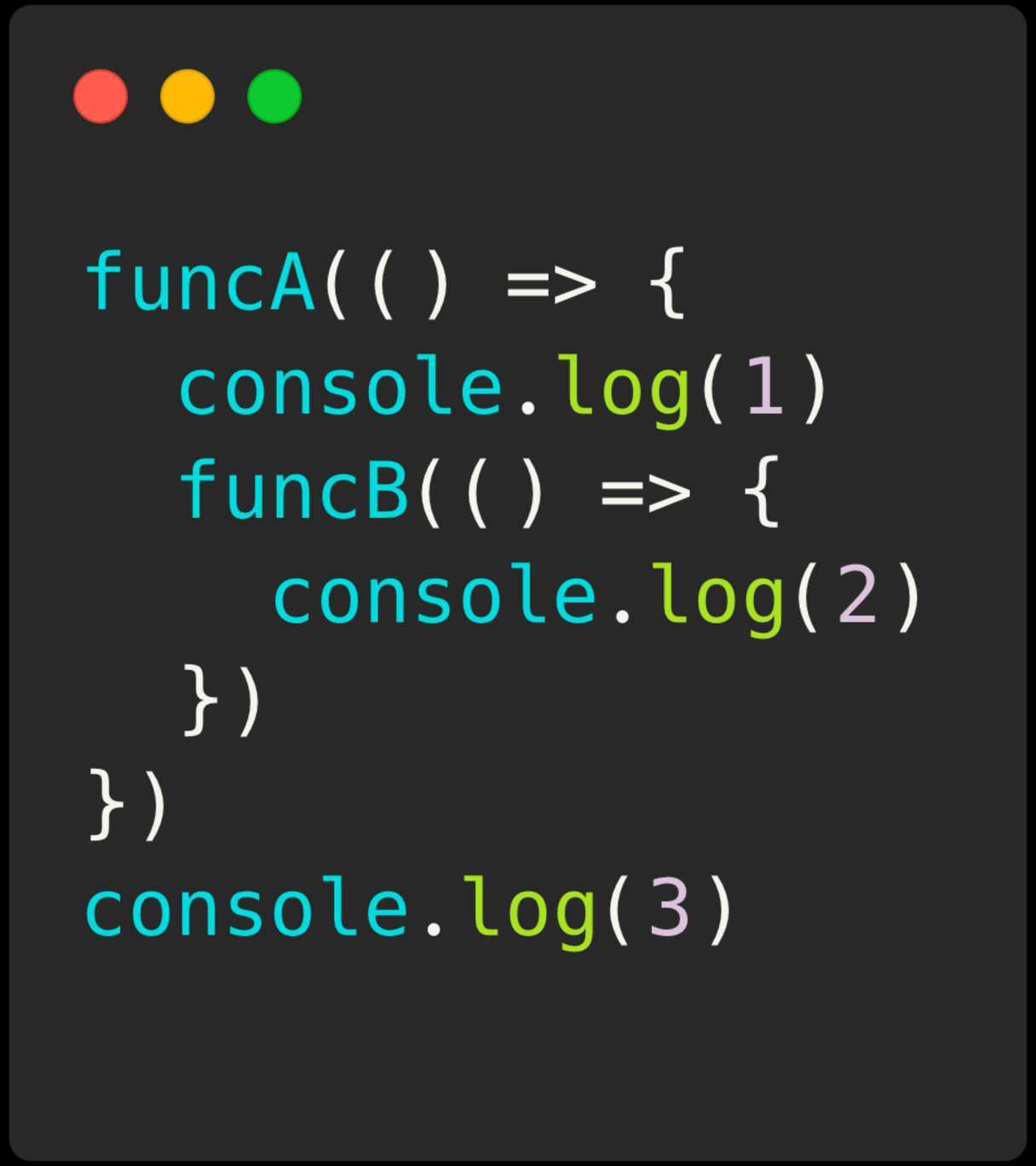
1. prototype chain 的概念
2. JS 透過 prototype chain 尋找屬性跟方法
3. 透過全寫去呼叫函式

Async

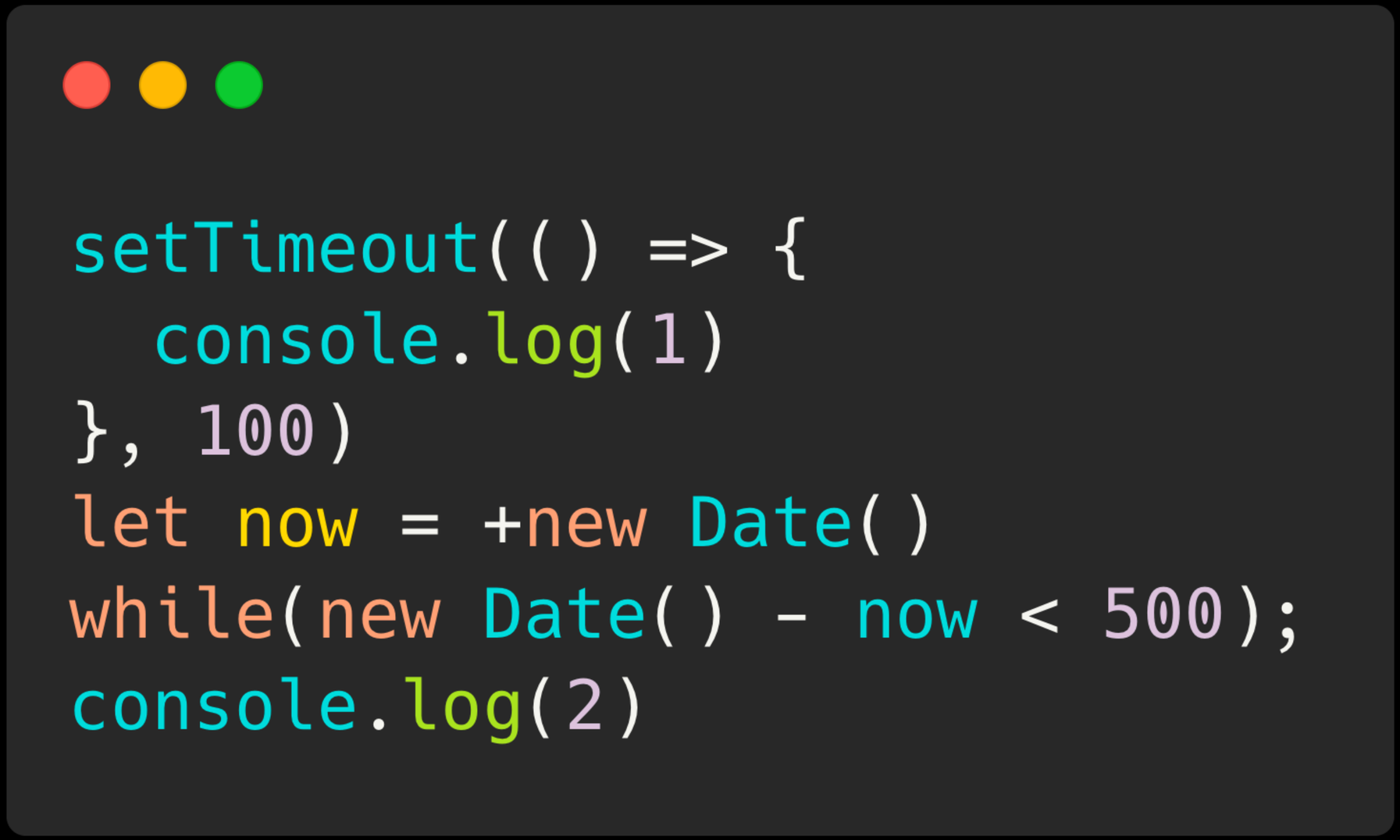

funcA(() => {
console.log(1)
funcB(() => {
console.log(2)
})
})
console.log(3)

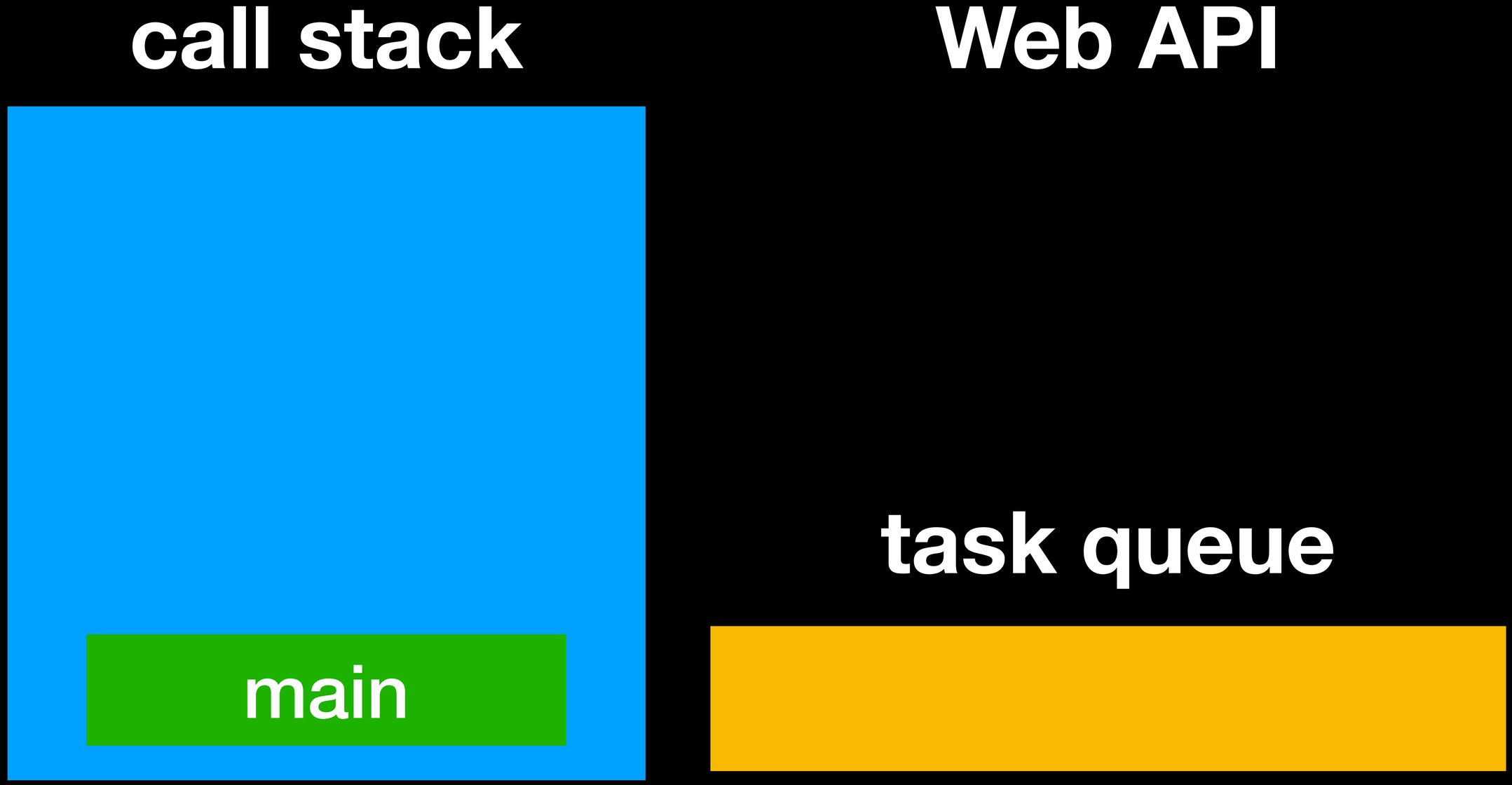


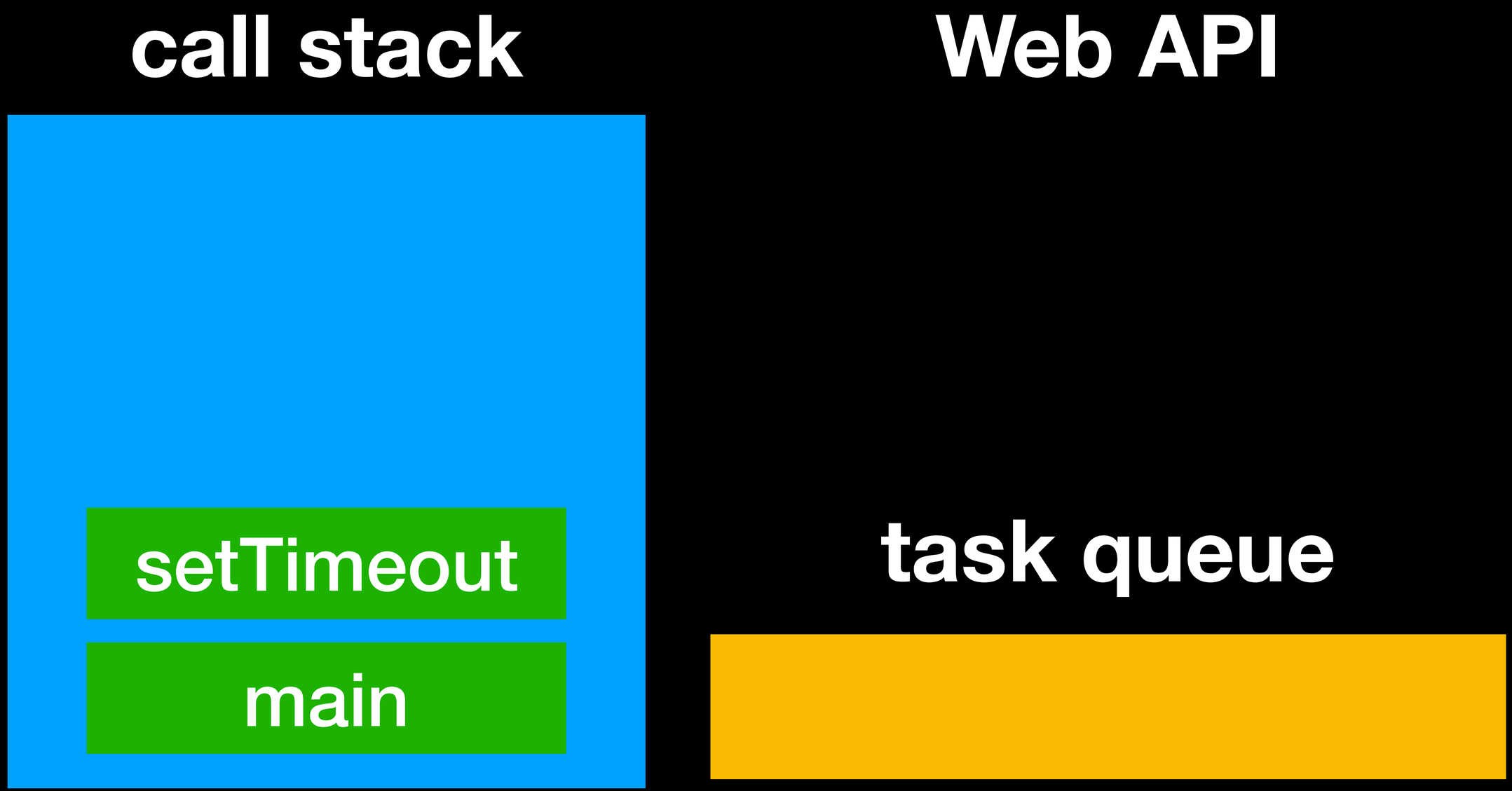
```
const run = fn => fn()
for(var i=1; i<=10; i++) {
  run(function() {
    console.log(i)
  })
}
```

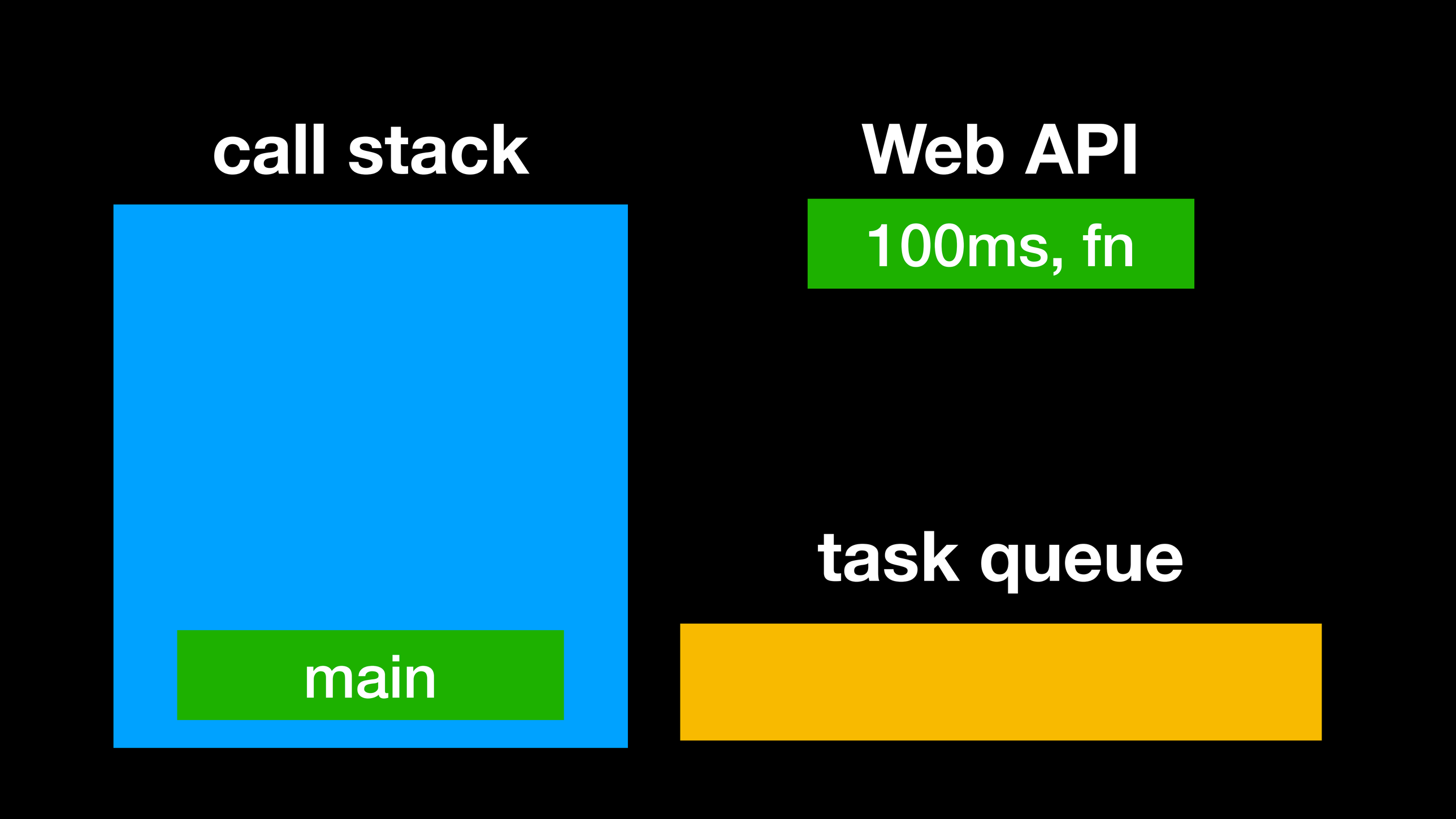

funcA(() => {
console.log(1)
funcB(() => {
console.log(2)
})
})
console.log(3)

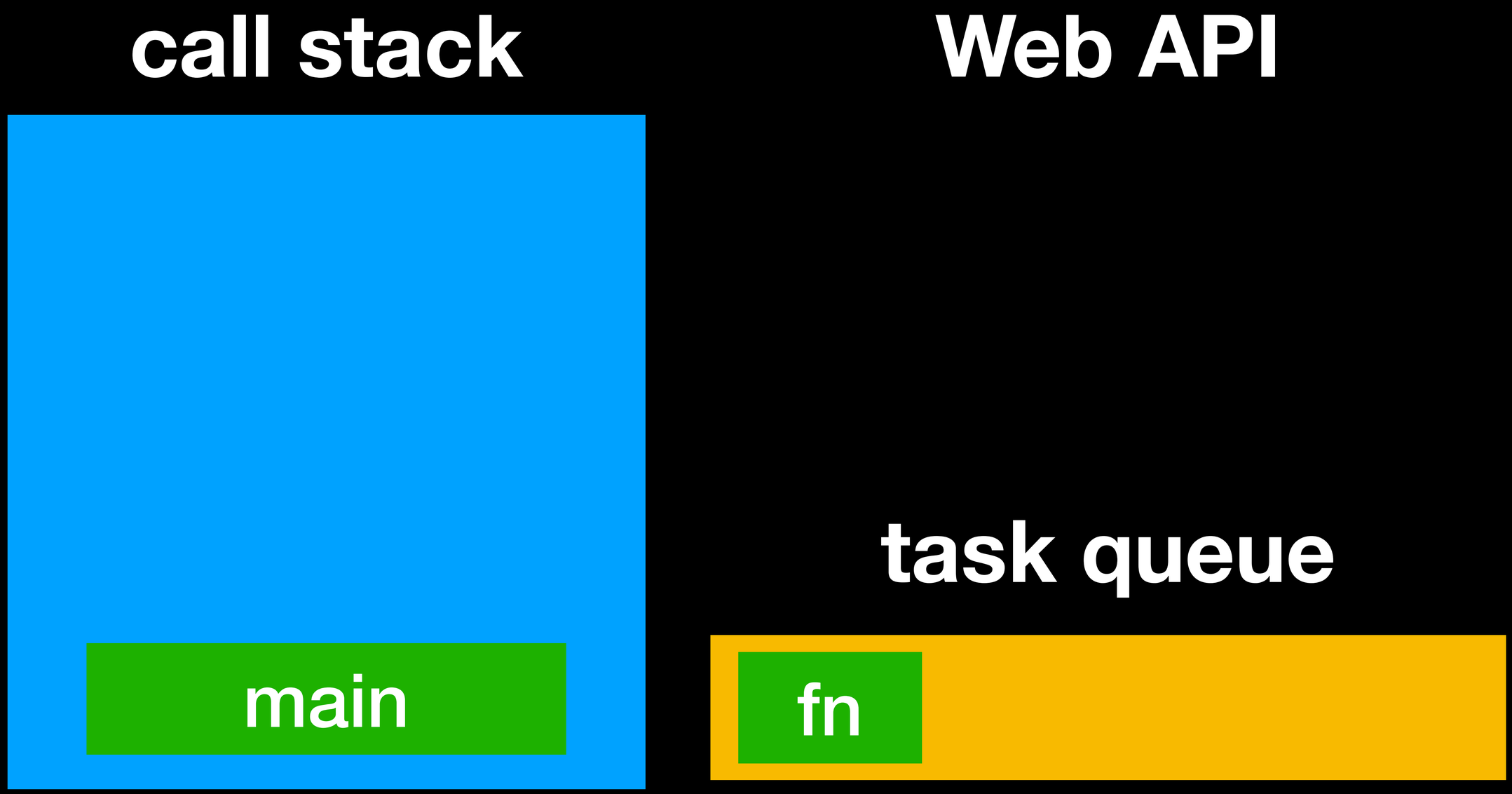
callback !== 非同步

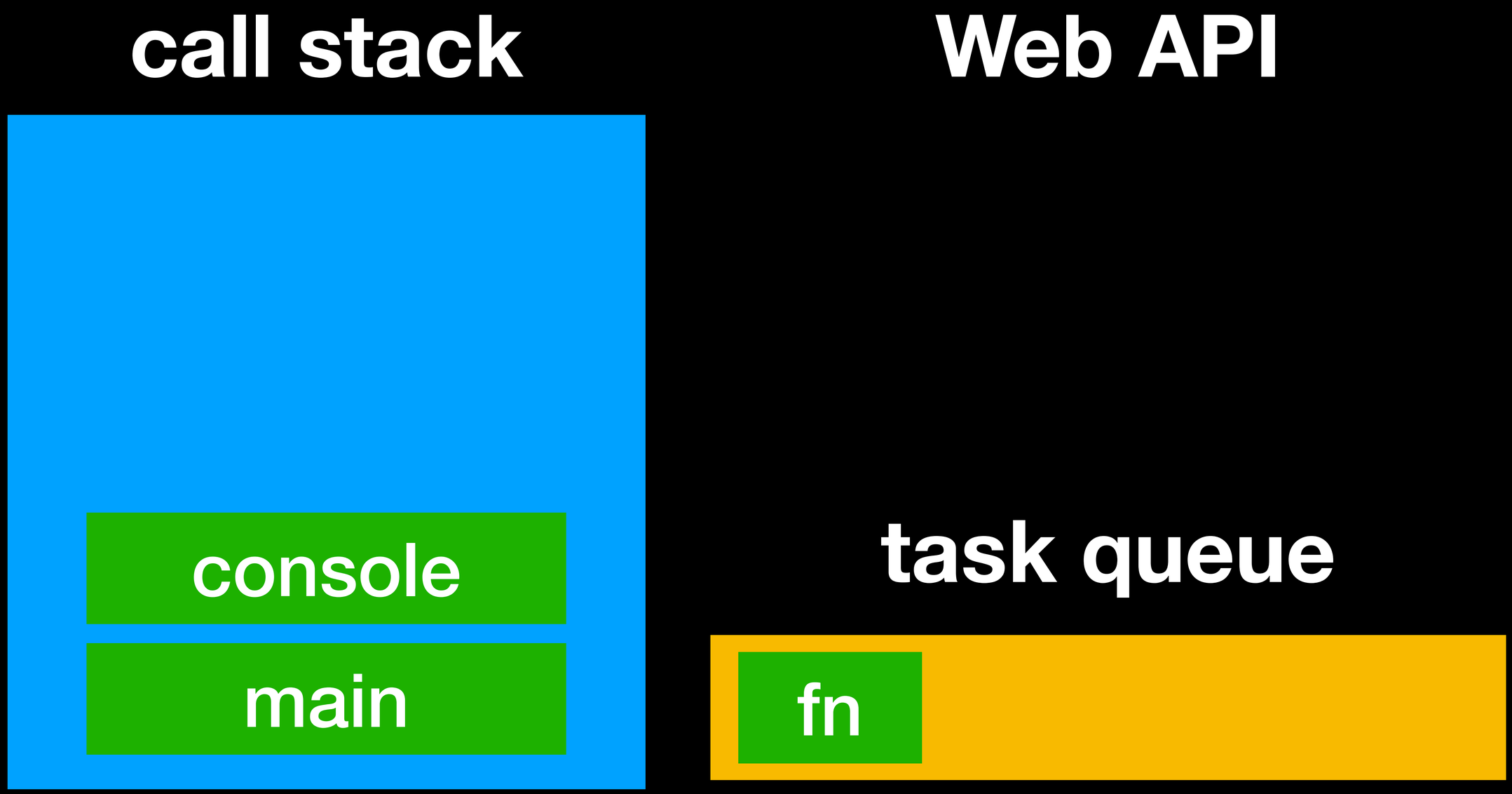

setTimeout(() => {
console.log(1)
}, 100)
let now = +new Date()
while(new Date() - now < 500);
console.log(2)

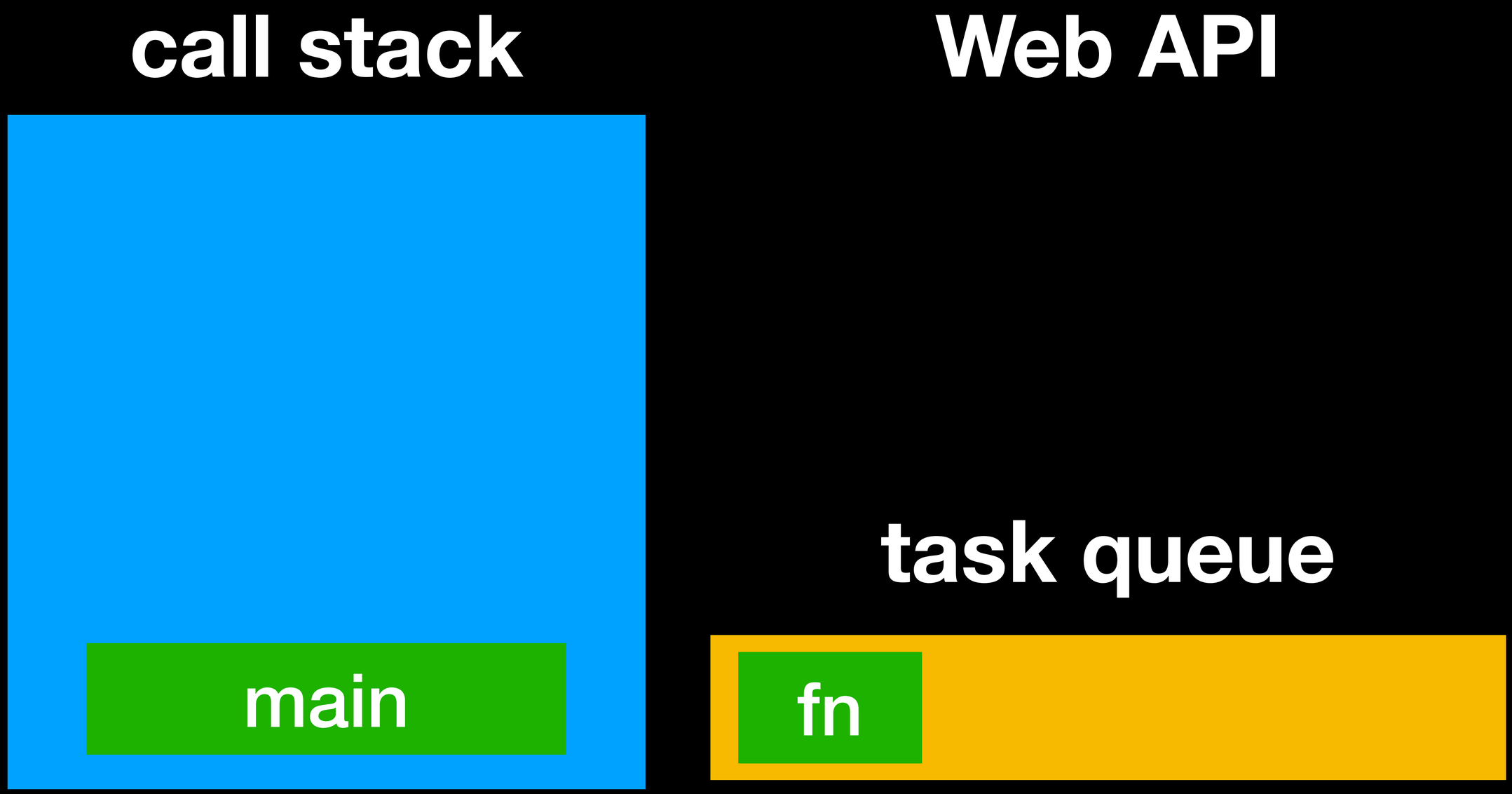

call stack
Web API
task queue
main


call stack
Web API
setTimeout
main
task queue


call stack
main
Web API
100ms, fn
task queue


call stack
Web API
task queue
main
fn

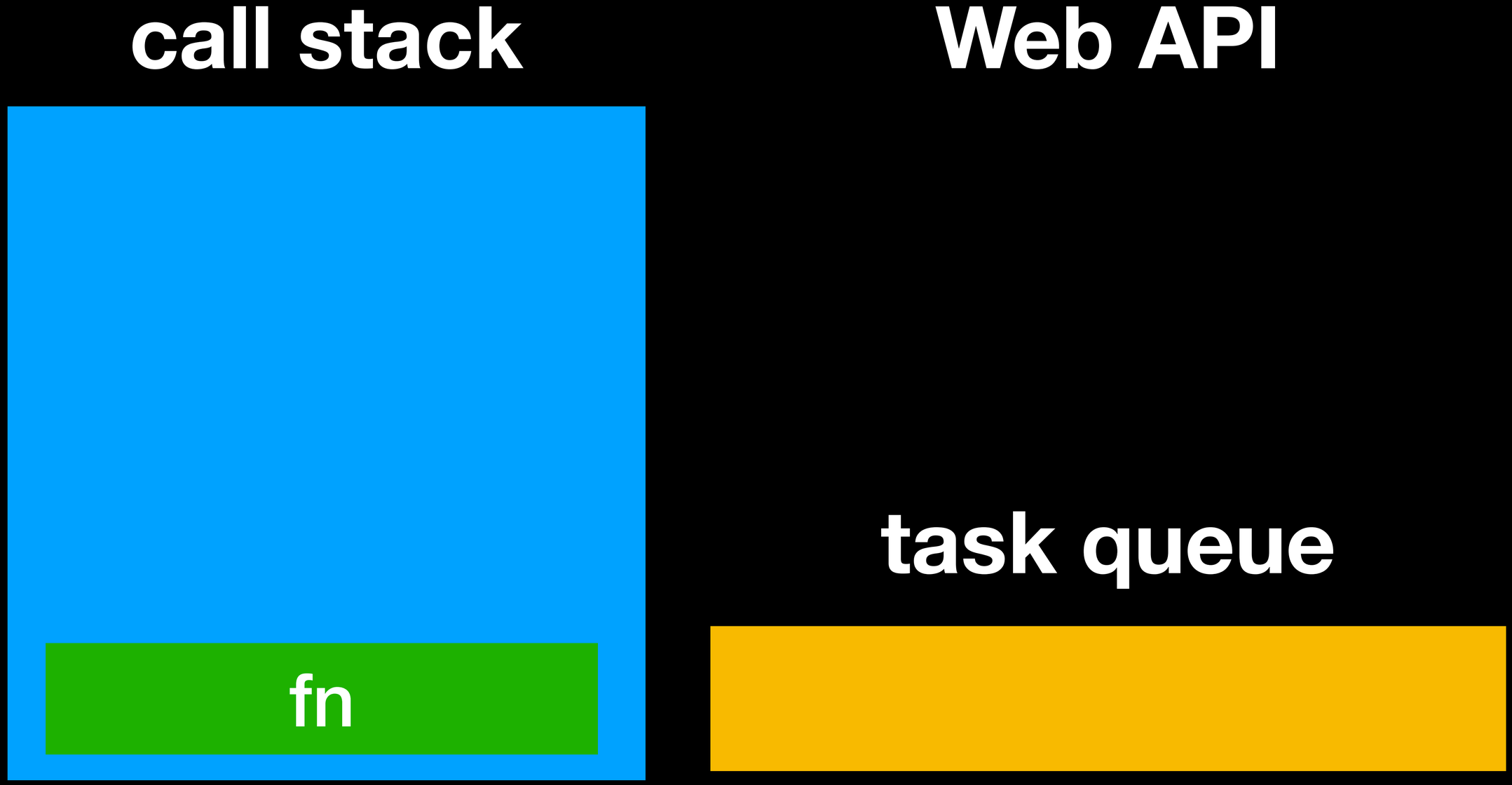

call stack
Web API
console
main
task queue
fn

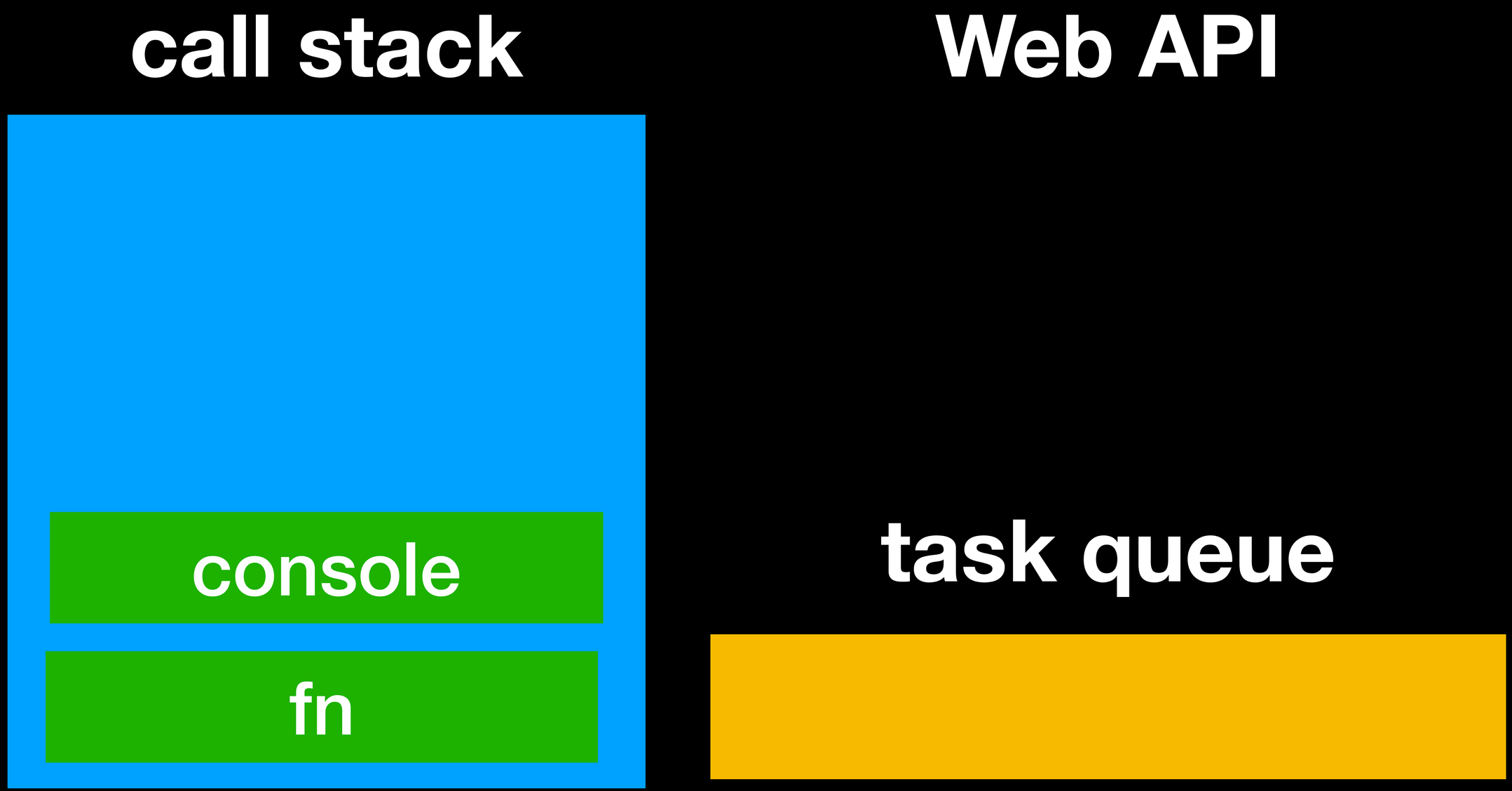

call stack
Web API
task queue
main
fn

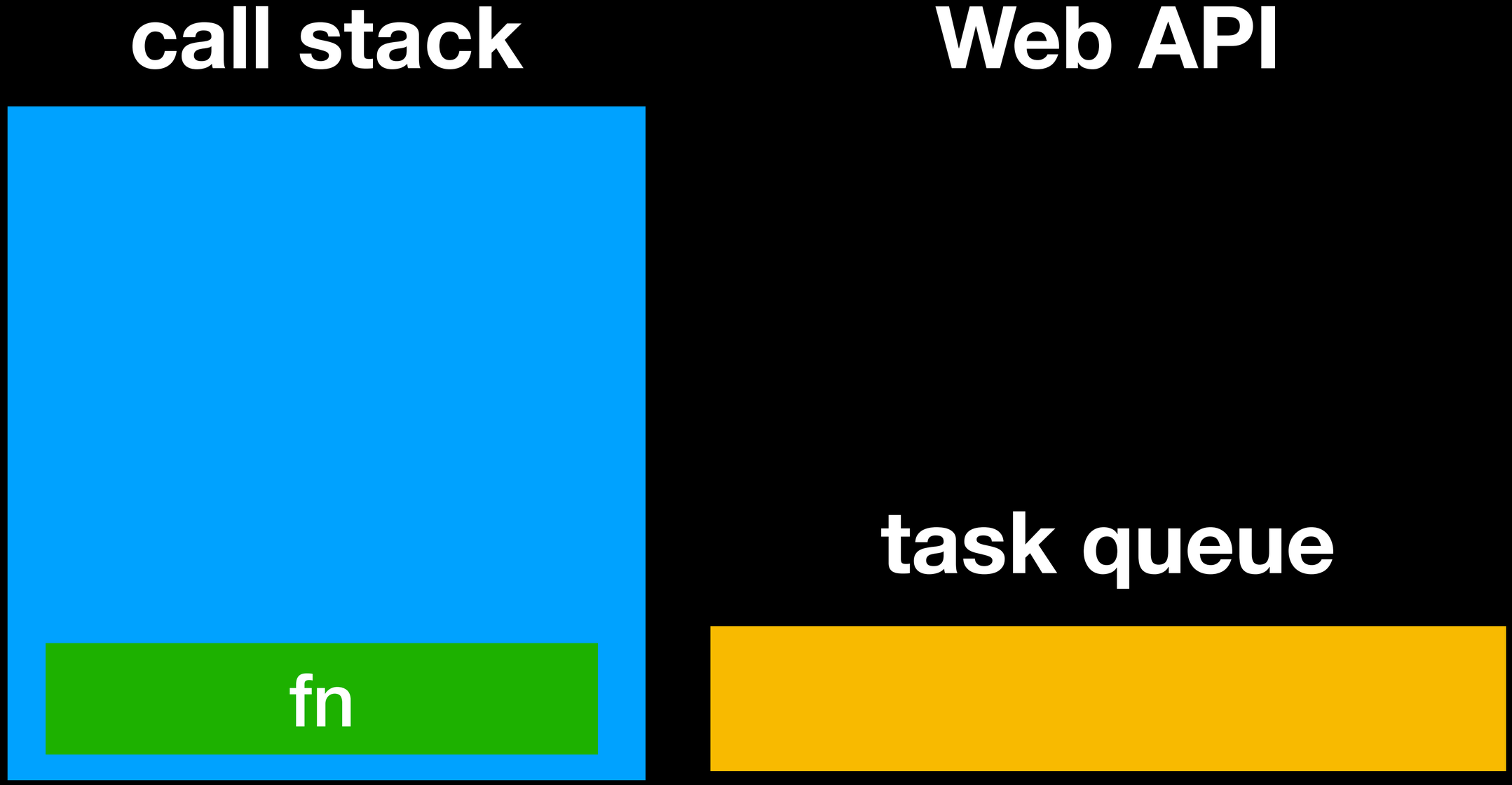
call stack

Web API

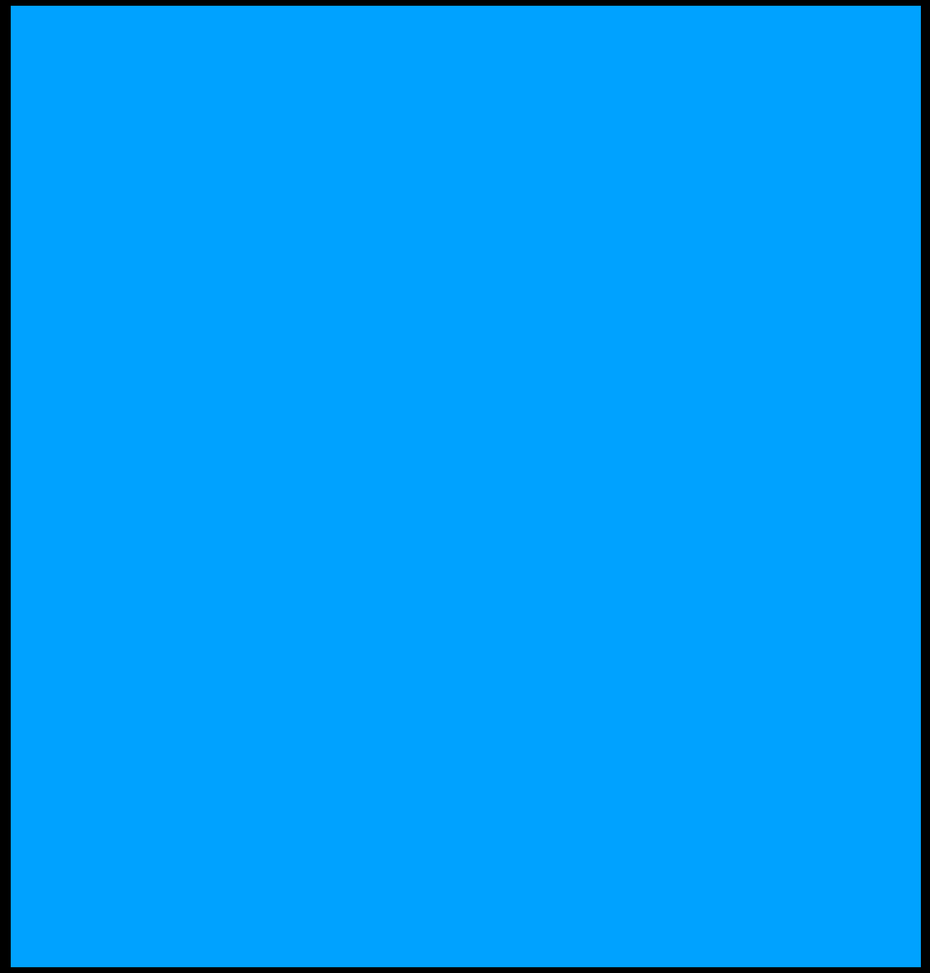
task queue


call stack
Web API
task queue
fn


call stack
Web API
console
fn
task queue

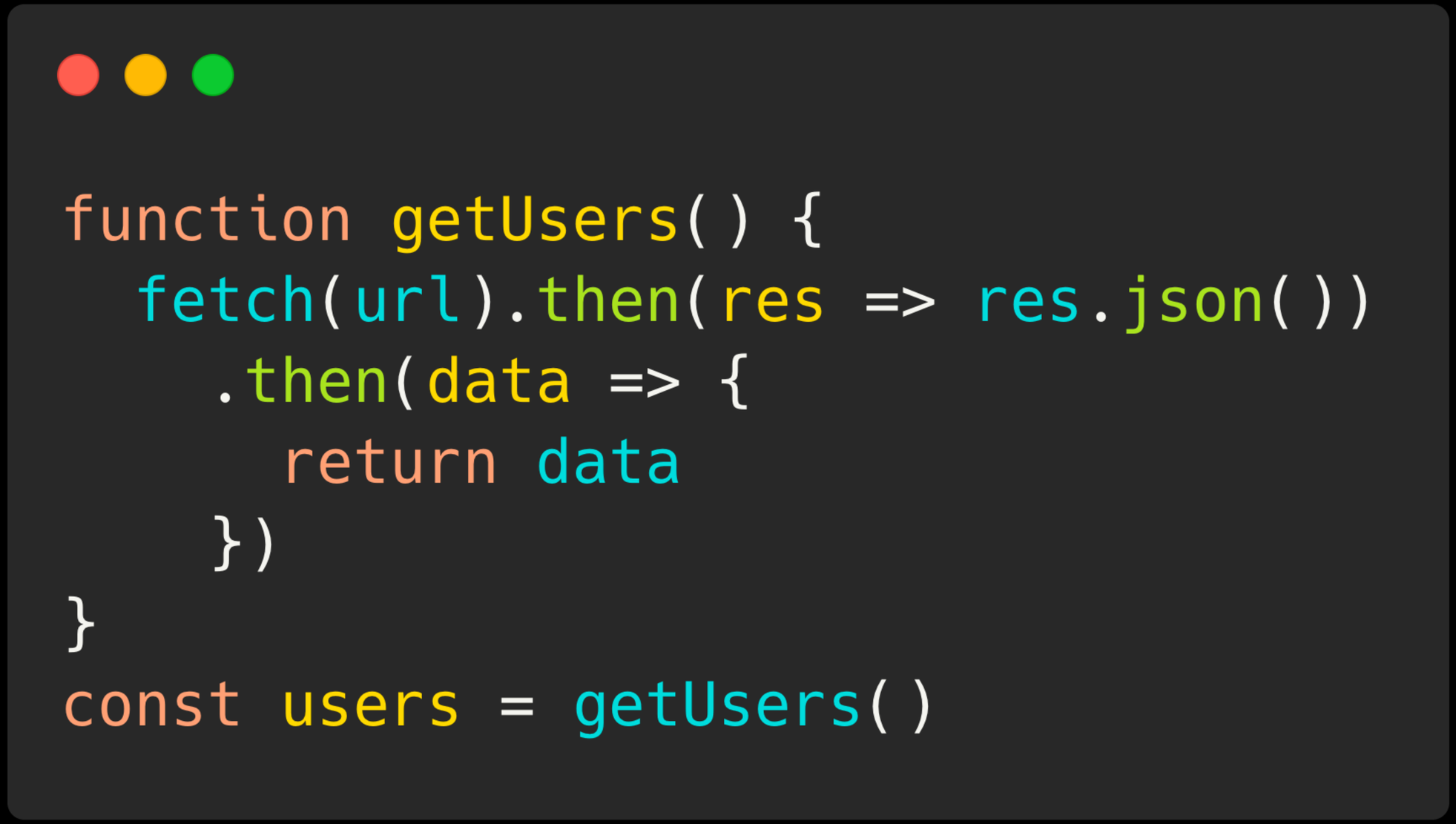

call stack
Web API
task queue
fn

call stack

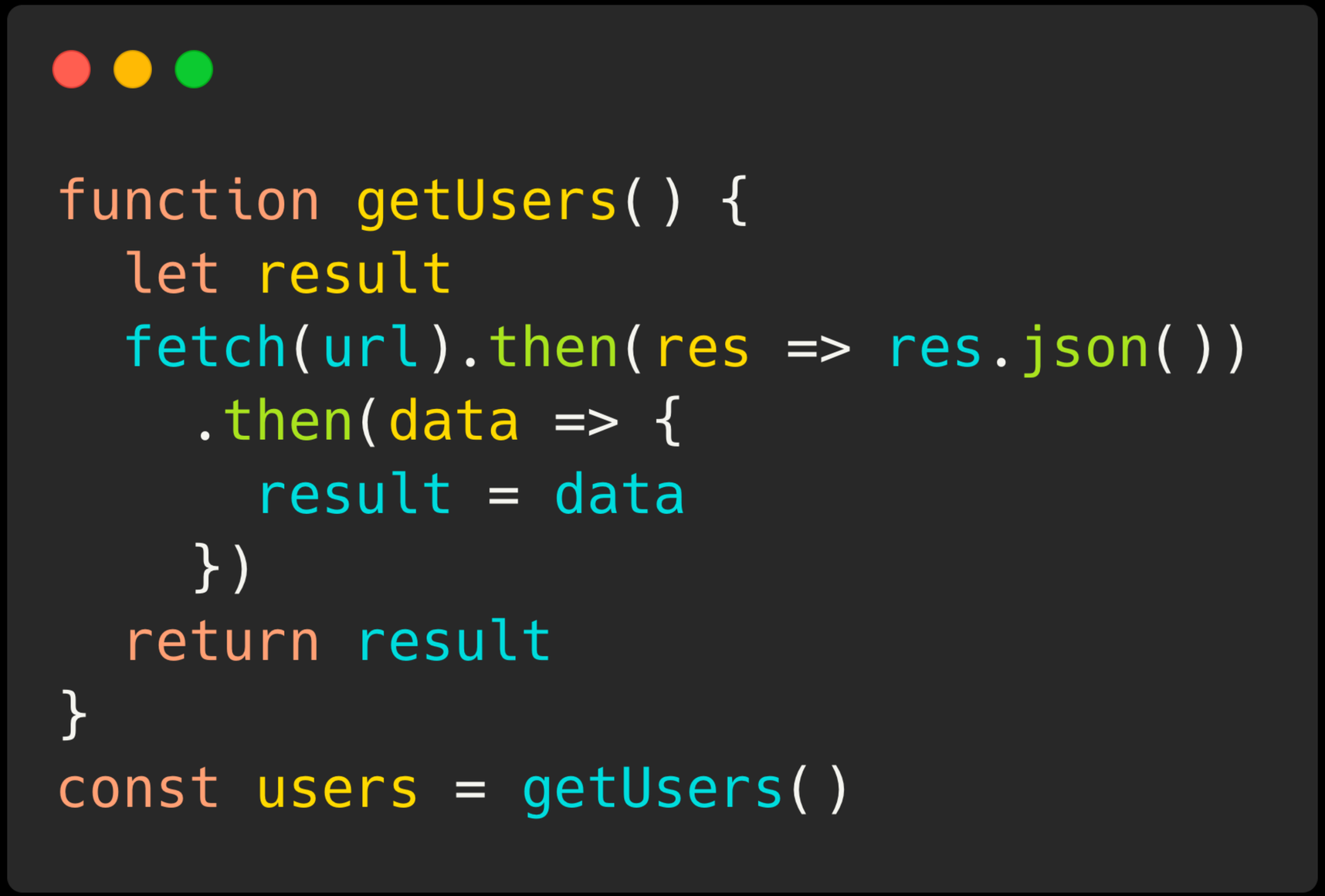


Web API

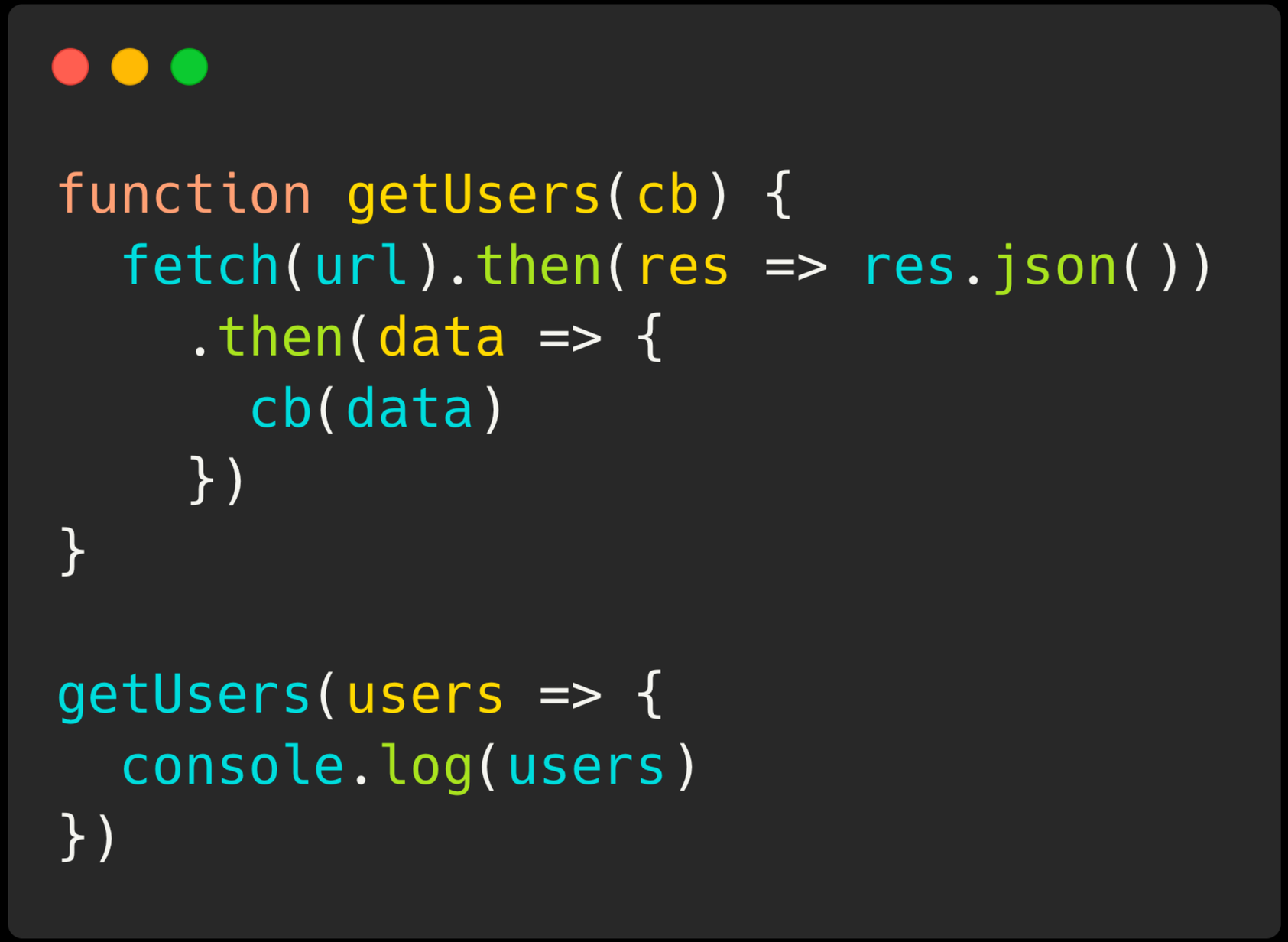
task queue

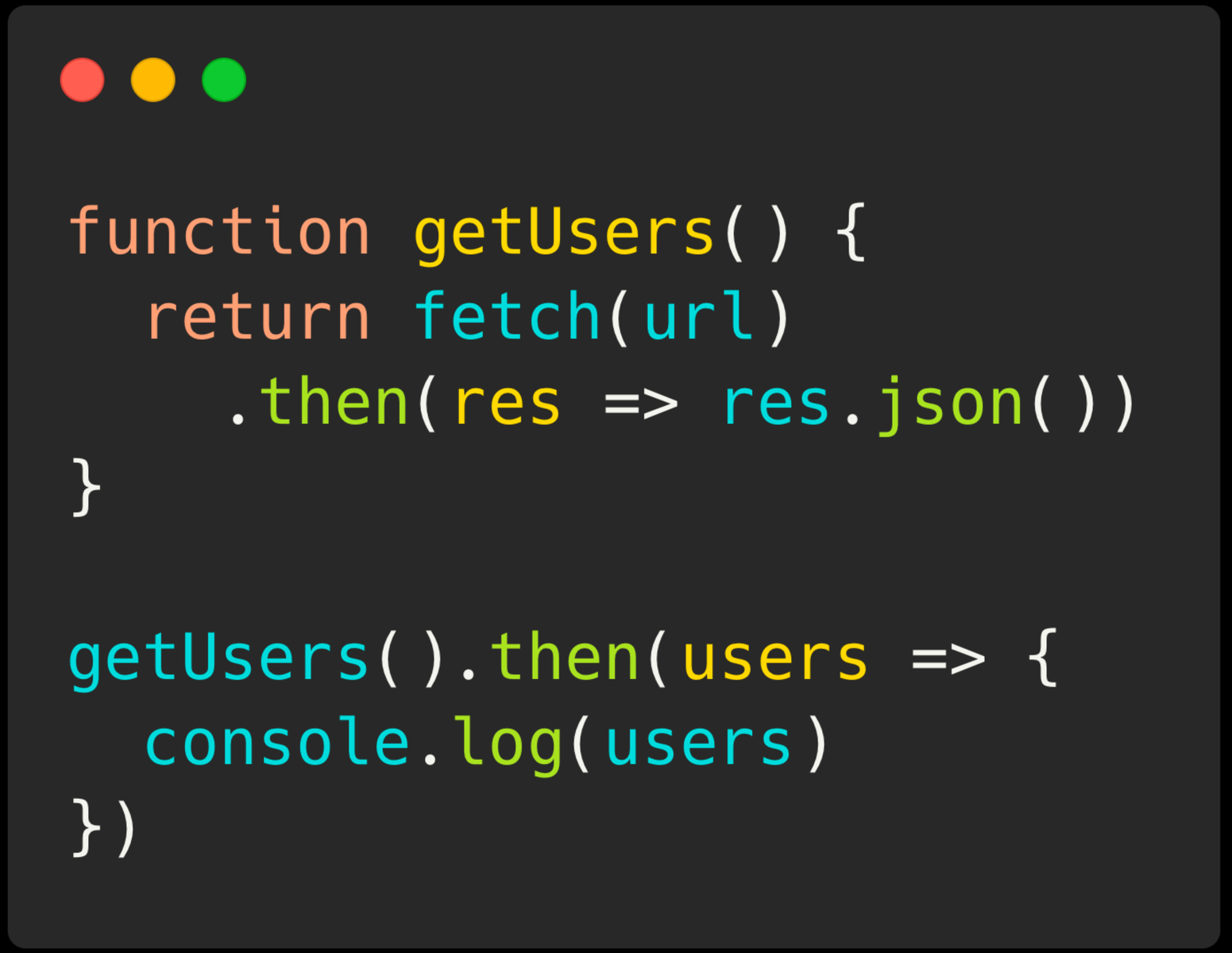


```
function getUsers() {
  fetch(url).then(res => res.json())
    .then(data => {
      return data
    })
}
const users = getUsers()
```



```
function getUsers() {
  let result
  fetch(url).then(res => res.json())
    .then(data => {
      result = data
    })
  return result
}
const users = getUsers()
```


function getUsers(cb) {
fetch(url).then(res => res.json())
.then(data => {
cb(data)
})
}
getUsers(users => {
console.log(users)
})


function getUsers() {
return fetch(url)
.then(res => res.json())
}
getUsers().then(users => {
console.log(users)
})

重點

1. callback !== 非同步
2. event loop 的運作方式

Agenda

1. Type
2. Hoisting
3. This
4. Prototype
5. Async

1. 這個對我寫 code 有幫助
2. 不知道的話，會容易寫出 bug