

# Node.js

# Objetivos del tema

- Entender la filosofía detrás de node.js
- Para qué sirve node.js
- Aprender a interactuar con la plataforma
- Aprender a interactuar con servicios externos

# **Antes de empezar**

# Requisitos

- editor de texto
- **node.js** versión 10
- familiaridad con **Javascript**
- *paciencia*

# **Introducción**

# Lenguaje disponible

- Node.js **10.1.0** soporta:
  - **ES2015**: 99%
  - **ES2016**: 100%
  - **ES2017**: 100%
  - **ES2018**: 100%
  - <https://node.green/>

# Fundamentos

- **Node.js** = ejecutar **javascript** en el **servidor**
  - ¿Por qué?

```
console.log('Hola, Mundo!')
```



# Fundamentos

- Para **ejecutar código**:
  - escribe el código en un fichero
  - ejecuta el comando:

```
node fichero.js
```

# Fundamentos

- Para **depurar código**:
  - puntos de ruptura con **debugger**
  - ejecuta el comando:

```
node inspect fichero.js
```

```
console.log('Hola, Mundo!')
```

```
let a = 1
```

```
function suma(a, b) {  
  return a + b  
}
```

```
debugger
```

```
console.log('> a:', a)  
suma(1, 1)  
console.log('adios')
```

```
console.log('Hola, Mundo!')
```

```
let a = 1
```

```
function suma(a, b) {  
  return a + b  
}
```

debugger

```
console.log('> a:', a)  
suma(1, 1)  
console.log('adios')
```

```
$ node inspect test.js
< Debugger listening on ws://127.0.0.1:9229/61a499ea
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in test.js:1
> 1 (function (exports, require, module, __filename, __dirname) {
  console.log('hola')
    2 let a = 1
    3 function suma(a, b) {
debug>
```

```
$ node inspect test.js
```

```
< Debugger listening on ws://127.0.0.1:9229/61a499ea
```

```
< For help, see: https://nodejs.org/en/docs/inspector
```

```
< Debugger attached.
```

```
Break on start in test.js:1
```

```
> 1 (function (exports, require, module, __filename, __dirname) {  
  console.log('hola')
```

```
  2 let a = 1
```

```
  3 function suma(a, b) {
```

```
debug>
```

```
$ node inspect test.js
```

```
< Debugger listening on ws://127.0.0.1:9229/61a499ea
```

```
< For help, see: https://nodejs.org/en/docs/inspector
```

```
< Debugger attached.
```

```
Break on start in test.js:1
```

```
> 1 (function (exports, require, module, __filename, __dirname) {
```

```
  console.log('hola')
```

```
    2 let a = 1
```

```
    3 function suma(a, b) {
```

```
debug>
```

```
$ node inspect test.js
```

```
< Debugger listening on ws://127.0.0.1:9229/61a499ea
```

```
< For help, see: https://nodejs.org/en/docs/inspector
```

```
< Debugger attached.
```

```
Break on start in test.js:1
```

```
> 1 (function (exports, require, module, __filename, __dirname) {  
  console.log('hola')
```

```
  2 let a = 1
```

```
  3 function suma(a, b) {
```

```
debug>
```



```
$ node inspect test.js
```

```
< Debugger listening on ws://127.0.0.1:9229/61a499ea
```

```
< For help, see: https://nodejs.org/en/docs/inspector
```

```
< Debugger attached.
```

```
Break on start in test.js:1
```

```
> 1 (function (exports, require, module, __filename, __dirname) {  
  console.log('hola')
```

```
    2 let a = 1
```

```
    3 function suma(a, b) {
```

```
debug>
```

# Comandos del depurador

- `list(n)`
  - Muestra n líneas alrededor
- `repl`
  - Arranca el intérprete de node
- `pause`
  - Interrumpe la ejecución

# Comandos del depurador

- **cont, c**
  - Continúa la ejecución
- **next, n**
  - Ejecuta una línea (con salto)
- **step, s**
  - Ejecuta una línea (sin salto)
- **out, o**
  - Ejecuta hasta escapar del contexto

# Comandos del depurador

- Depurar con DevTools
  - `node --inspect-brk test.js`
  - abre Chrome
  - ve a la url `chrome://inspect`
  - deberías ver el depurador escuchando
  - haz click en inspect

# Errores

- Cuando se lanza una **excepción** y no se captura...
  - El proceso termina
  - Se muestra un resumen de la excepción en la consola

```
function ping(n = 0) {  
  if (n > 5) throw new Error('Oh, noes!');  
  return pong(n)  
}
```

```
function pong(n) {  
  return ping(n + 1)  
}
```

```
ping()
```

```
$ node error.js
  /path/to/error.js:2
if (n > 5) throw new Error('Oh, noes!');
  ^
```

```
Error: Oh, noes!
at ping (/path/to/error.js:2:20)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
```

```
$ node error.js
```

```
  /path/to/error.js:2  
if (n > 5) throw new Error('Oh, noes!');  
  ^
```

Error: Oh, noes!

at ping (/path/to/error.js:2:20)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)



```
$ node error.js
  /path/to/error.js:2
if (n > 5) throw new Error('Oh, noes!');
  ^
```

Error: Oh, noes!

```
at ping (/path/to/error.js:2:20)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
```

```
$ node error.js
  /path/to/error.js:2
if (n > 5) throw new Error('Oh, noes!');
  ^
```

Error: Oh, noes!

```
at ping (/path/to/error.js:2:20)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
```

# Ejercicio

- Utiliza el depurador para encontrar el **bug** en el este programa...

# Ejercicio

```
function createFns() {  
  let fns = []  
  for (var i = 0; i < 10; i++) {  
    fns.push(function() { console.log(i) })  
  }  
  return fns  
}
```

```
var fns = createFns()  
fns[0] // ??  
fns[1] // ??
```

# Módulos

# Módulos

- Cada fichero es **un módulo independiente**
- Se ejecuta en un **ámbito aislado**
- Puede **importar** otros módulos
- Puede **exportar** nombres

# Módulos

- El código de un módulo se **ejecuta** cuando:
  - Lo pasamos como parámetro al invocar a **node**
  - **La primera vez** que es importado

colors.js

---

```
exports.red = '#d30139'  
exports.green = '#01d339'  
exports.blue = '#0139d3'
```



colors.js

---

```
exports.red = '#d30139'  
exports.green = '#01d339'  
exports.blue = '#0139d3'
```

index.js

---

```
const colors = require('./colors.js')  
console.log('Red:', colors.red)
```

index.js

---

```
const colors = require('./colors.js')  
console.log('Red:', colors.red)
```

index.js

---

```
const colors = require('./colors.js')  
console.log('Red:', colors.red)
```

# Módulos

- Dentro del ámbito del módulo tenemos 4 **variables**
  - **module**: *referencia al módulo actual*
  - **require**: *función para importar otros módulos*
  - **\_\_filename**: *string con la ruta del fichero del módulo*
  - **\_\_dirname**: *string con la ruta del directorio del módulo*

index.js

---

```
console.log(`Gracias por ejecutar ${__filename}`)
```

# Módulos

- Los módulos solo se ejecutan **una vez**
  - Se cachea todo lo que haya exportado
  - La siguiente vez que se requiera, se utilizan los valores cacheados

module.js

---

console.log( *'Running!'* )



index.js

---

```
require('./module.js')  
require('./module.js')  
require('./module.js')  
require('./module.js')
```

# Módulos

- La ruta que pasamos a **require**...
  - Si empieza con **/** es una ruta absoluta
  - Si empieza con **./** o **../** es una ruta relativa
  - En caso contrario, ruta relativa a **\$NODE\_PATH**
    - `node_modules`

# Módulos

- Si la ruta **es un directorio**....
  - node busca en el interior del directorio
  - intenta cargar el fichero **index.js**
  - si no existe, levanta un error

# Módulos

- **require** también puede importar **ficheros .json**
  - Un array u objeto por fichero
  - Ese array u objeto será el valor importado

# Ejercicio

- Crea un fichero **filters.js** que exporte dos funciones:
  - **isSeniorUser(user)**
  - **isAcmeEmployee(user)**

# Ejercicio

- Crea un script **index.js** que...
  - Lea el fichero **exercises/1-node/01/users.json**
  - Cargue las funciones de **filters.js**
  - Muestre por consola...
    - los nombres de los **senior users**
    - el número de usuarios empleados de **ACME**

module.js

---

```
console.log(module.exports) // {}  
exports.prop = 1  
console.log(module.exports) // { prop: 1 }
```

module.js

---

```
module.exports.otherProp = 2  
console.log(exports.otherProp) // 2  
console.log(module.exports === exports) // true
```



module.js

---

```
module.exports = { prop: 'oh, noes!' }  
console.log(module.exports === exports) // FALSE!
```

# Módulos

- La expresión **require(...)**
  - devuelve el valor de **module.exports** del módulo importado
  - habitualmente es **un objeto**
  - pero puede ser **cualquier valor**

module.js

---

module.exports = *'Something else'*

index.js

---

```
const what = require('./module.js')  
console.log(what) // Something else
```

# Ejercicio

- Reescribe **filters.js** para que...
  - utilice **module.exports** en lugar de **exports**
  - exporte un objeto con las dos funciones
- Modifica **index.js** para que cargue las funciones correctamente

**npm**

# npm

- **N**ode **P**ackage **M**anager
- `package.json`
- Múltiples usos:
  - gestor de dependencias
  - configuración del proyecto
  - comandos de compilación, testing y despliegue

# npm

- Inicializamos un proyecto con: **npm init**
  - Un asistente para crear la configuración inicial
  - Genera el fichero **package.json**



```
{  
  "name": "test",  
  "version": "1.0.0",  
  "description": "Just a test package",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha ./tests"  
  },  
  "repository": {  
    "type": "git",  
    "url": "http://www.github.com/test/test"  
  },  
  "keywords": [  
    "test"  
  ],  
  "author": "Elias Alonso",  
  "license": "ISC"  
}
```

```
{  
  "name": "test",  
  "version": "1.0.0",  
  "description": "Just a test package",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha ./tests"  
  },  
  "repository": {  
    "type": "git",  
    "url": "http://www.github.com/test/test"  
  },  
  "keywords": [  
    "test"  
  ],  
  "author": "Elias Alonso",  
  "license": "ISC"  
}
```

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Just a test package",
  "main": "index.js",
  "scripts": {
    "test": "mocha ./tests"
  },
  "repository": {
    "type": "git",
    "url": "http://www.github.com/test/test"
  },
  "keywords": [
    "test"
  ],
  "author": "Elias Alonso",
  "license": "ISC"
}
```

```
{  
  "name": "test",  
  "version": "1.0.0",  
  "description": "Just a test package",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha ./tests"  
  },  
  "repository": {  
    "type": "git",  
    "url": "http://www.github.com/test/test"  
  },  
  "keywords": [  
    "test"  
  ],  
  "author": "Elias Alonso",  
  "license": "ISC"  
}
```

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Just a test package",
  "main": "index.js",
  "scripts": {
    "test": "mocha ./tests"
  },
  "repository": {
    "type": "git",
    "url": "http://www.github.com/test/test"
  },
  "keywords": [
    "test"
  ],
  "author": "Elias Alonso",
  "license": "ISC"
}
```

# npm

- Los “scripts” son comandos que se pueden ejecutar...
  - `npm run <script>`
  - útil para comandos de complicación, testing, despliegue....
  - `node_modules/.bin` está en el `$PATH`

# npm

- Podemos **instalar paquetes** con...
  - `npm install <paquete>`
  - el paquete se convierte en una *dependencia*
  - se añade a `package.json`

```
$ npm install lodash
```



```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Just a test package",
  "main": "index.js",
  "scripts": {
    "test": "mocha ./tests"
  },
  "repository": {
    "type": "git",
    "url": "http://www.github.com/test/test"
  },
  "author": "Elias Alonso",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.10"
  }
}
```

```
$ ls  
node_modules  
package.json  
package-lock.json
```

```
project
|
├─ node_modules
├─ package-lock.json
├─ package.json
├─ public
├─ src
|   └─ index.js
└─ tests
```

# Ejercicio

- Inicializa un proyecto con **npm init**
- Instala los paquetes **lodash** y **nodemon**
- Monta la estructura de directorios sugerida
- Crea un script “*start*” que ejecute **src/index.js**
- Crea un script “*dev*” que ejecute **src/index.js** con **nodemon**

**testing**

# testing

- **Jest**

- Herramienta moderna y completa
- Poca configuración
- Muchos *features* integrados
  - matchers
  - espías
  - cobertura

```
$ npm install jest
```

# testing

- Nuestra **metodología**:
  - Código en **src/fichero.js**
  - Tests en **test/fichero.test.js**



```
const { reverseString } = require('../src/utils.js')

describe('reverseString(str)', () => {

  it('should not change a one char string', () => {
    expect(reverseString('o')).toBe('o')
  })

})
```

```
const { reverseString } = require('../src/utils.js')
```

```
describe('reverseString(str)', () => {
```

```
  it('should not change a one char string', () => {  
    expect(reverseString('o')).toBe('o')  
  })
```

```
})
```

```
const { reverseString } = require('../src/utils.js')
```

```
describe('reverseString(str)', () => {
```

```
  it('should not change a one char string', () => {  
    expect(reverseString('o')).toBe('o')  
  })
```

```
})
```

```
const { reverseString } = require('../src/utils.js')

describe('reverseString(str)', () => {

  it('should not change a one char string', () => {
    expect(reverseString('o')).toBe('o')
  })

})
```

```
const { reverseString } = require('../src/utils.js')

describe('reverseString(str)', () => {

  it('should not change a one char string', () => {
    expect(reverseString('o')).toBe('o')
  })

})
```

```
const { reverseString } = require('../src/utils.js')

describe('reverseString(str)', () => {

  it('should not change a one char string', () => {
    expect(reverseString('o')).toBe('o')
  })

})
```

```
const { reverseString } = require('../src/utils.js')

describe('reverseString(str)', () => {

  it('should not change a one char string', () => {
    expect(reverseString('o')).toBe('o')
  })

})
```

\$ jest



```
$ jest --watchAll
```

# Ejercicio

- Utiliza **jest** para...
  - Programar la función **reverseString(...)**
  - Utilizando **TDD**

# testing

- Jest tiene un montón de **matchers**
  - <https://jestjs.io/docs/en/expect>
  - Ve a echar un vistazo
  - Conocer los matchers ayuda a escribir tests más legibles y sucintos

```
const { isAcmeEmployee } = require('../src/filters.js')

describe('isAcmeEmployee', () => {

  let users

  beforeEach(() => {
    users = [
      { name: 'Test', email: 'test@acme.com', age: 28 },
      { name: 'Test', email: 'test@nop.com', age: 32 }
    ]
  })

  it('should return true when the email ends in acme.com', () => {
    expect(isAcmeEmployee(users[0])).toBe(true)
  })
})
```

```
const { isAcmeEmployee } = require('../src/filters.js')
```

```
describe('isAcmeEmployee', () => {
```

```
  let users
```

```
    beforeEach(() => {  
      users = [  
        { name: 'Test', email: 'test@acme.com', age: 28 },  
        { name: 'Test', email: 'test@nop.com', age: 32 }  
      ]  
    })
```

```
    it('should return true when the email ends in acme.com', () => {  
      expect(isAcmeEmployee(users[0])).toBe(true)  
    })  
  })
```

# testing

- **beforeEach, afterEach**

- Ejecutar una función antes o después de **cada test**

- **beforeAll, afterAll**

- Ejecutar una función **una vez** antes o después de ejecutar los tests

- Scope limitado al bloque **describe** en el que aparecen

```
describe('Async code', () => {  
  it('should wait for the timeout', (done) => {  
    setTimeout(() => {  
      expect(true).toBe(true)  
      done()  
    }, 1000)  
  })  
})
```

```
describe('Async code', () => {  
  it('should wait for the timeout', (done) => {  
    setTimeout(() => {  
      expect(true).toBe(true)  
      done()  
    }, 1000)  
  })  
})
```



```
describe('Async code', () => {  
  it('should wait for the timeout', (done) => {  
    setTimeout(() => {  
      expect(true).toBe(true)  
    }, 1000)  
  })  
})
```

```
describe('Async code', () => {  
  it('should wait for the timeout', (done) => {  
    setTimeout(() => {  
      expect(true).toBe(true)  
      done()  
    }, 1000)  
  })  
})
```

# Ejercicio

- Utiliza **jest** para...
  - Testear la función **throttle(ms, fn)**
  - `exercises/1-node/03`

# testing

- Jest ofrece **mock functions**
  - Funciones “falsas”
  - Para controlar invocaciones y valores de retorno

```
const myFn = jest.fn()
```

```
myFn(1)
```

```
myFn('Hello', 'World')
```

```
console.log(myFn.mock.calls) // [[1], ['Hello', 'World']]
```

```
const myFn = jest.fn()
```

```
myFn(1)
```

```
myFn('Hello', 'World')
```

```
console.log(myFn.mock.calls) // [[1], ['Hello', 'World']]
```

# Ejercicio

- Reescribe los test de **throttle...**
  - Utilizando **mock functions**

# **concurrency**



# Concurrencia

- Node.js ejecuta nuestro código en **una sola hebra**
  - Por tanto, **NO HAY CONCURRENCIA**

```
console.log('empezamos')  
  
for (let i=1e10; i--;)  
  void 0  
  
console.log('terminamos')
```



```
console.log('empezamos')
```

```
for (let i=1e10; i--;)
```

```
    void 0
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```



```
for (let i=1e10; i--:)
```

```
void 0
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```

```
for (let i=1e10; i--;)
  void 0
```

```
 console.log('terminamos')
```

```
console.log('empezamos')  
  
for (let i=1e10; i--;)   
    void 0  
  
console.log('terminamos')
```



# Concurrencia

- Node.js ejecuta nuestro **código** en **una sola hebra**
  - Por tanto, **NO HAY CONCURRENCIA**
- Node.js ejecuta las operaciones de **I/O** en **hebras distintas**
  - El código js **no** es concurrente
  - La **plataforma si es concurrente!**



```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



```
console.log('empezamos')
```



```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```



```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



*Cuando hayan pasado 1000ms...*

```
console.log('empezamos')
```

➡ setTimeout(() => console.log('cuándo?'), 1000)

```
console.log('terminamos')
```

*Cuando hayan pasado 1000ms...*

console.log('cuándo?')

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

➡ console.log('terminamos')

*Cuando hayan pasado 1000ms...*

```
console.log('cuándo?')
```



```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



*Cuando hayan pasado 1000ms...*

```
console.log('cuándo?')
```



```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



*Cuando hayan pasado 1000ms...*



```
console.log('cuándo?')
```

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



*Cuando hayan pasado 1000ms...*

```
console.log('cuándo?')
```





➡ console.log('empezamos')  
setTimeout(() => console.log('cuánto tarda?'), 1)  
for (let i=1e10; i--;) void 0  
console.log('terminamos')

```
console.log('empezamos')
```

➡ setTimeout(() => console.log('cuánto tarda?'), 1)

```
for (let i=1e10; i--;) void 0
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```

➡ `setTimeout(() => console.log('cuánto tarda?'), 1)`

```
for (let i=1e10; i--;) void 0
```

```
console.log('terminamos')
```

Cuando haya pasado 1ms...

```
console.log('cuánto tarda?')
```

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuánto tarda?'), 1)
```



```
for (let i=1e10; i--;) void 0
```

```
console.log('terminamos')
```



*Cuando haya pasado 1ms...*

```
console.log('cuánto tarda?')
```



```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuánto tarda?'), 1)
```

➡ for (let i=1e10; i--;) void 0

```
console.log('terminamos')
```

*Ya ha pasado 1ms!*

```
console.log('cuánto tarda?')
```

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuánto tarda?'), 1)
```

```
for (let i=1e10; i--;) void 0
```

➡ console.log('terminamos')

*Ya han pasado muchos ms!*

```
console.log('cuánto tarda?')
```

```
console.log('empezamos')  
  
setTimeout(() => console.log('cuánto tardo?'), 1)  
  
for (let i=1e10; i--;) void 0  
  
console.log('terminamos')
```



*Ya han pasado muchos ms!*

```
console.log('cuánto tardo?')
```



```
console.log('empezamos')  
  
setTimeout(() => console.log('cuánto tarda?'), 1)  
  
for (let i=1e10; i--;) void 0  
  
console.log('terminamos')
```

*Ya han pasado muchos ms!*

➡ console.log('cuánto tarda?')

```
console.log('empezamos')  
  
setTimeout(() => console.log('cuánto tarda?'), 1)  
  
for (let i=1e10; i--;) void 0  
  
console.log('terminamos')
```

*Ya han pasado muchos ms!*

```
console.log('cuánto tarda?')
```



# Concurrencia


- Node.js ejecuta nuestro código en **una sola hebra**
  - Solo podemos ejecutar **una cosa cada vez**
  - La **plataforma** es **concurrente**
  - Nuestro **código NO**

# Concurrencia

- Es una **decisión de diseño**
  - Para **simplificar** la escritura de código
  - La visión: miles de clientes **simultáneos** con código menos propenso a errores y más fácil de mantener
  - Mantener la concurrencia “oculta”
  - Increíblemente eficaz en casos donde **I/O > cpu**



```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```




```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

*Cuando hayan pasado 1ms...*



```
console.log('uno')
```




```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

*Cuando hayan pasado 1ms...*

```
console.log('uno')
```

*Cuando hayan pasado 1ms...*

```
console.log('dos')
```



```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

*Cuando hayan pasado 1ms...*

```
console.log('uno')
```


*Cuando hayan pasado 1ms...*

```
console.log('dos')
```

*Cuando hayan pasado 1ms...*

```
console.log('tres')
```





```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

*Cuando hayan pasado 1ms...*

```
console.log('uno')
```

*Cuando hayan pasado 1ms...*

```
console.log('dos')
```

*Cuando hayan pasado 1ms...*

```
console.log('tres')
```

```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

 Cuando hayan pasado 1ms...

```
console.log('uno')
```

Cuando hayan pasado 1ms...

```
console.log('dos')
```

Cuando hayan pasado 1ms...

```
console.log('tres')
```



```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

*Ya ha pasado 1 ms!*



```
console.log('uno')
```

*Ya ha pasado 1 ms!*

```
console.log('dos')
```

*Ya ha pasado 1 ms!*

```
console.log('tres')
```

```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

*Ya ha pasado 1 ms!*

```
console.log('uno')
```

*Ya ha pasado 1 ms!*

➡ console.log('dos')

*Ya ha pasado 1 ms!*

```
console.log('tres')
```

```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

*Ya ha pasado 1 ms!*

```
console.log('uno')
```

*Ya ha pasado 1 ms!*

```
console.log('dos')
```

*Ya ha pasado 1 ms!*

```
console.log('tres')
```

```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

*Ya ha pasado 1 ms!*

```
console.log('uno')
```

*Ya ha pasado 1 ms!*

```
console.log('dos')
```

*Ya ha pasado 1 ms!*

```
console.log('tres')
```

# Concurrencia

- El **patrón** es siempre el mismo:
  - El **código principal** arranca una serie de operaciones
  - Con bloques de código asociados
  - Y **termina!**
    - *El programa se ejecuta de arriba a abajo sin parar!*
    - *Tiene que terminar para que la hebra se quede libre*



# Concurrencia

- **NUNCA BLOQUEAR LA HEBRA!**
  - No podemos responder a los eventos asíncronos!
  - El programa se queda “colgado”

# Concurrencia

- Configuramos una **condición**
- A la que asociamos **una función**
- Que **no ejecutamos nosotros**
- **Delegamos su ejecución** en la plataforma
- Y **alteramos el flujo natural del programa**
  - ejecución **desordenada**

# Concurrencia

- El *intérprete* ejecuta **el código** en una sola hebra...
- ...pero **cada proceso de I/O** tiene **su propia hebra!**
- ...y se comunican con la hebra principal **mediante callbacks**

# Callbacks

# Callbacks

- Un **callback** es
  - Una **función**
  - Que **definimos nosotros**
  - Pero que **será ejecutada** por **otro agente**
  - Con **parámetros** que describen el suceso que al que estaba asociado

```
const callback = () => alert('hi');  
  
setTimeout(callback, 100);
```

# Callbacks

- **TODA** la asincronía en JS se basa en callbacks
  - las demás técnicas son patrones sobre callbacks
- Mecanismo de “**bajo nivel**”
- Continuation Passing Style (CPS)

# Callbacks

- **DOS** limitaciones importantes respecto a las funciones
  - **NO** se puede recuperar su *valor de retorno*
  - **NO** se pueden capturar sus *excepciones*



```
function delaySum(a, b) {  
  setTimeout(() => a + b, 100);  
}
```

```
const result = delaySum(12, 90);  
console.log(result); // ???
```

# Callbacks

- La **única** manera de “devolver” un valor desde un callback es **llamando a otro callback**
  - La asincronía es *contagiosa*
  - En cuanto un valor es asíncrono, **todo el código que lo utilice** va a ser asíncrono también

```
function delaySum(a, b, callback) {  
  setTimeout(() => callback(a + b), 100);  
}
```

```
delaySum(12, 90, (result) => {  
  console.log(result);  
});
```

```
function delaySum(a, b, callback) {  
  setTimeout(() => callback(a + b), 100);  
}
```

```
delaySum(12, 90, (result) => {  
  console.log(result);  
});
```

```
function delaySum(a, b, callback) {  
  setTimeout(() => callback(a + b), 100);  
}
```

```
delaySum(12, 90, (result) => {  
  console.log(result);  
});
```

# Ejercicio

- Escribe una función **tirarDado(ncaras, cb)**
  - Simula el lanzamiento de un dado de n caras
  - Tarda un número aleatorio de **ms** entre **100** y **500**
  - Pasa el resultado como primer parámetro a **cb**
  - **exercises/1-node/04**
  - **TDD**

# Ejercicio

```
tirarDado(6, (result) => {  
  console.log(result) // 2  
})
```

```
tirarDado(10, (result) => {  
  console.log(result) // 5  
})
```

```
function delayDiv(a, b, callback) {  
  setTimeout(() => {  
    if (b === 0) throw new Error('Div by 0!');  
    callback(a / b);  
  }, 100);  
}  
  
try {  
  delayDiv(12, 0, (result) => {  
    console.log(result);  
  });  
} catch(e) {  
  console.log('Safely captured:', e.message);  
}
```



# Callbacks

- El **acuerdo general** (sobre todo en node.js) es:
  - Los callbacks **NUNCA** levantan excepción
  - Si hay algún error, se pasa **como primer parámetro** al **callback**
  - Si todo va bien, el primer parámetro se deja a **null**

```
function delayDiv(a, b, callback) {
  setTimeout(() => {
    if (b === 0) {
      callback(new Error('Div by 0!'))
    } else {
      callback(null, a / b);
    }
  }, 100);
}

delayDiv(12, 0, (err, result) => {
  if (err) {
    console.log('Safely captured:', err.message);
  } else {
    console.log(result);
  }
});
```

```
function delayDiv(a, b, callback) {
  setTimeout(() => {
    if (b === 0) {
      callback(new Error('Div by 0!'))
    } else {
      callback(null, a / b);
    }
  }, 100);
}

delayDiv(12, 0, (err, result) => {
  if (err) {
    console.log('Safely captured:', err.message);
  } else {
    console.log(result);
  }
});
```

# Ejercicio

- Escribe una función **tirarDados(listaCaras, cb)**
  - Recibe un **array** de caras
  - Llama a **tirarDado** con *cada elemento* del array
  - Llama a **cb** con los resultados **de todos los dados!**
    - cuando haya *terminado todas las llamadas*
  - **TDD**

# Ejercicio

```
tirarDados([4, 4, 6, 6], (results) => {  
  console.log(results) // [2, 1, 4, 3]  
})
```

**plataforma**

# Plataforma

- Node.js es una plataforma **back-end**
  - no tenemos **interfaz de usuario**
  - **interacción** con **el sistema**
  - y sus **servicios**
  - escribiendo **scripts** o **servidores**

# Plataforma

- Node.js no es sólo el intérprete de js
  - también es **la librería** estándar
  - para interactuar con el **sistema**
  - de una **manera peculiar**
  - para **sacar provecho** de su **modelo de concurrencia**



# Plataforma: módulo “fs”

- Interactuar con el **sistema de ficheros**
  - crear, leer, modificar y borrar ficheros
  - listar directorios
  - consultar detalles de un fichero
  - cambiar permisos
  - .....

```
const fs = require('fs')

fs.readdir('.', (err, files) => {
  if (err)
    console.error(err)
  else
    console.log(files)
})

console.log('antes o después?')
```

```
const fs = require('fs')
```

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

```
const fs = require('fs')
```

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

```
const fs = require('fs')
```

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

```
const fs = require('fs')
```

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

```
const fs = require('fs')
```

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

```
const fs = require('fs')
```

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```



```
const fs = require('fs')

fs.readdir('.', (err, files) => {
  if (err)
    console.error(err)
  else
    console.log(files)
})
```

```
console.log('antes o después?')
```

# Ejercicio

- Escribe un programa que...
  - reciba **un directorio** como parámetro
    - en la invocación por línea de comandos
  - y calcule recursivamente su **tamaño**

# Ejercicio

```
$ node index.js ~/Downloads/  
Total: 281.96 Mb
```

# Ejercicio

- **process.argv**
  - lista con los componentes de la invocación
- **fs.readdir**
  - lista los elementos de un directorio
- **path.join**
  - concatenar segmentos de una ruta

# Ejercicio

- **fs.stat**
  - consultar si una ruta es directorio o fichero
  - consultar el tamaño de un fichero

# Plataforma

Para testear un programa que utilice la plataforma, tenemos dos opciones...

- Falsear el `require` con `jest.mock`
- Inyectar los módulos desde fuera

# Plataforma

- **Jest.mock** nos permite...
  - capturar las llamadas a `require`
  - sustituir el módulo cargado por uno falso
  - implementar una funcionalidad de testing en el módulo falso

```
const fs = require('fs')
const calculateDirSize = require('../src/calculate-dir-size.js')

jest.mock('fs')

describe('calculate-dir-size', () => {
  it('should do nothing if empty directory', (done) => {
    fs.readdir.mockImplementation((_, cb) => cb(null, []))
    calculateDirSize('.', (err, result) => {
      expect(result).toBe(0)
      done()
    })
  })
})
```



```
const fs = require('fs')
```

```
const calculateDirSize = require('../src/calculate-dir-size.js')
```

```
jest.mock('fs')
```

```
describe('calculate-dir-size', () => {  
  it('should do nothing if empty directory', (done) => {  
    fs.readdir.mockImplementation((_ , cb) => cb(null, []))  
    calculateDirSize('.', (err, result) => {  
      expect(result).toBe(0)  
      done()  
    })  
  })  
})
```

```
const fs = require('fs')
const calculateDirSize = require('../src/calculate-dir-size.js')

jest.mock('fs')

describe('calculate-dir-size', () => {
  it('should do nothing if empty directory', (done) => {
    fs.readdir.mockImplementation((_, cb) => cb(null, []))
    calculateDirSize('.', (err, result) => {
      expect(result).toBe(0)
      done()
    })
  })
})
```

```
const fs = require('fs')
const calculateDirSize = require('../src/calculate-dir-size.js')

jest.mock('fs')

describe('calculate-dir-size', () => {
  it('should do nothing if empty directory', (done) => {
    fs.readdir.mockImplementation((_, cb) => cb(null, []))
    calculateDirSize('.', (err, result) => {
      expect(result).toBe(0)
      done()
    })
  })
})
```

```
const fs = require('fs')
const calculateDirSize = require('../src/calculate-dir-size.js')

jest.mock('fs')

describe('calculate-dir-size', () => {
  it('should do nothing if empty directory', (done) => {
    fs.readdir.mockImplementation((_, cb) => cb(null, []))
    calculateDirSize('.', (err, result) => {
      expect(result).toBe(0)
      done()
    })
  })
})
```

# Plataforma

- **Injectar módulos** nos ofrece...
  - una buena práctica
  - que no depende de ninguna herramienta externa
  - código más mantenible y flexible
  - la mejor de las dos opciones (cuando sea posible)

```
function calculateDirSize(fs, path, cb) {  
  fs.readdir(path, (err, files) => {  
    const fullPathFiles = files.map(f => `${path}/${f}`)  
    asyncMap(  
      fullPathFiles,  
      (path, cb) => calculateSize(fs, path, cb),  
      (err, sizes) => {  
        if (err) return cb(err)  
        const totalSize = sizes.reduce((acc, el) => acc + el, 0)  
        cb(null, totalSize)  
      })  
    )  
  })  
}
```

```
function calculateDirSize(fs, path, cb) {  
  fs.readdir(path, (err, files) => {  
    const fullPathFiles = files.map(f => `${path}/${f}`)  
    asyncMap(  
      fullPathFiles,  
      (path, cb) => calculateSize(fs, path, cb)  
    )(err, sizes) => {  
      if (err) return cb(err)  
      const totalSize = sizes.reduce((acc, el) => acc + el, 0)  
      cb(null, totalSize)  
    }  
  })  
}
```

```
const calculateDirSize = require('../src/calculate-dir-size.js')

describe('calculate-dir-size', () => {
  it('should do nothing if there directory is empty', (done) => {
    const fakeFs = { readdir: (_, cb) => cb(null, []) }
    calculateDirSize(fakeFs, '.', (err, result) => {
      expect(result).toBe(0)
      done()
    })
  })
})
```



```
const calculateDirSize = require('../src/calculate-dir-size.js')

describe('calculate-dir-size', () => {
  it('should do nothing if there directory is empty', (done) => {
    const fakeFs = { readdir: (_, cb) => cb(null, []) }
    calculateDirSize(fakeFs, '.', (err, result) => {
      expect(result).toBe(0)
      done()
    })
  })
})
```

# Ejercicio

- Escribe un programa que...
  - reciba **un directorio** como parámetro
    - en la invocación por línea de comandos
  - busque recursivamente un **substring** en el contenido de los ficheros
    - un clon de grep

# Ejercicio

```
$ node index.js "const" ~/code/  
./path1/file.js: const fs = require('fs')  
./path2/index.js: const next = (remaining)  
./path2/index.js: const [head, ...tail]
```

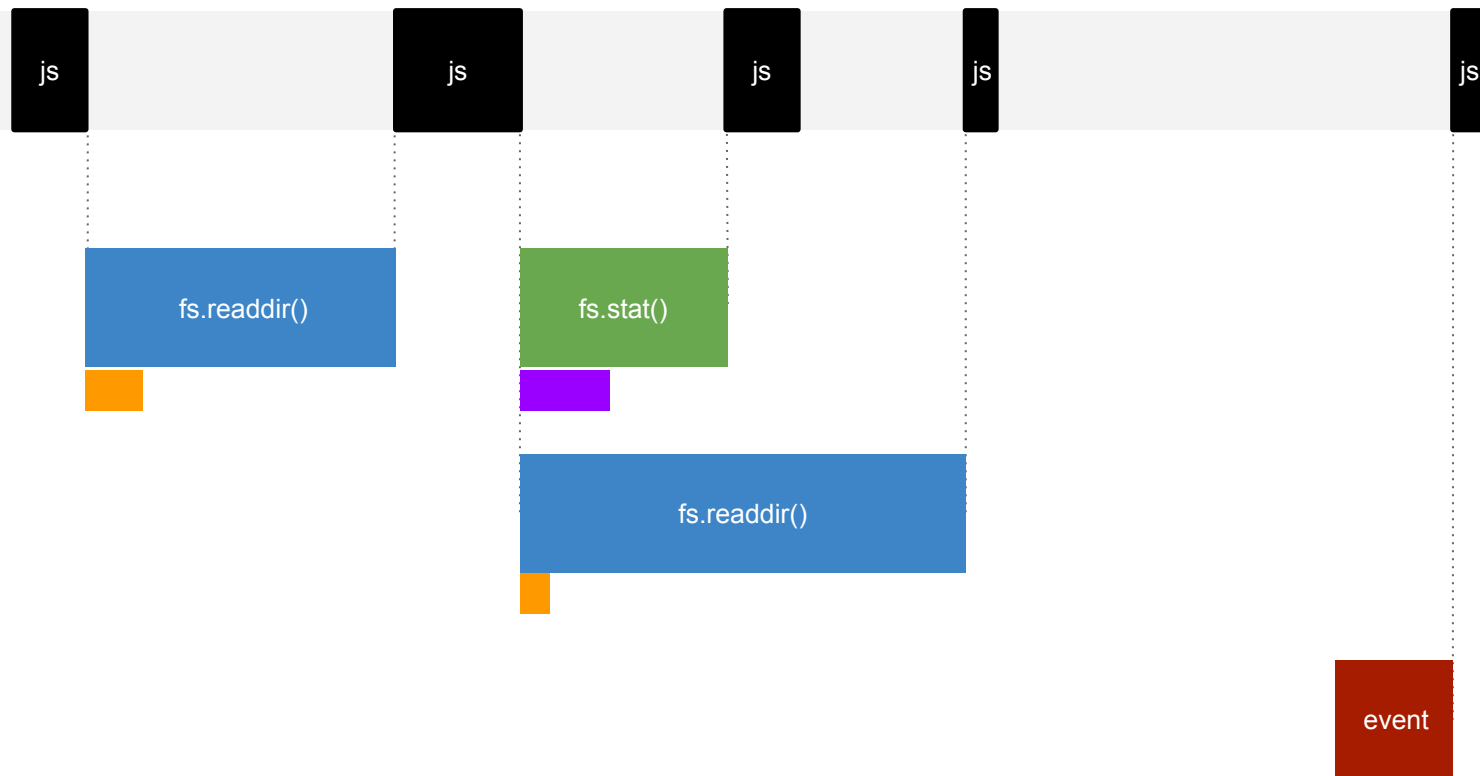
# Ejercicio

- **fs.readFile**
  - leer el contenido de un fichero

# **streams**

# Streams

- El modelo de operaciones asíncronas que hemos visto...
  - funciona muy bien para gestiones “pequeñas”



# Streams

- Este modelo de operaciones asíncronas...
  - funciona muy bien para gestiones “pequeñas”
  - pero es problemático para volúmenes más grandes
    - mucha memoria consumida
    - mucho tiempo de bloque de hebra



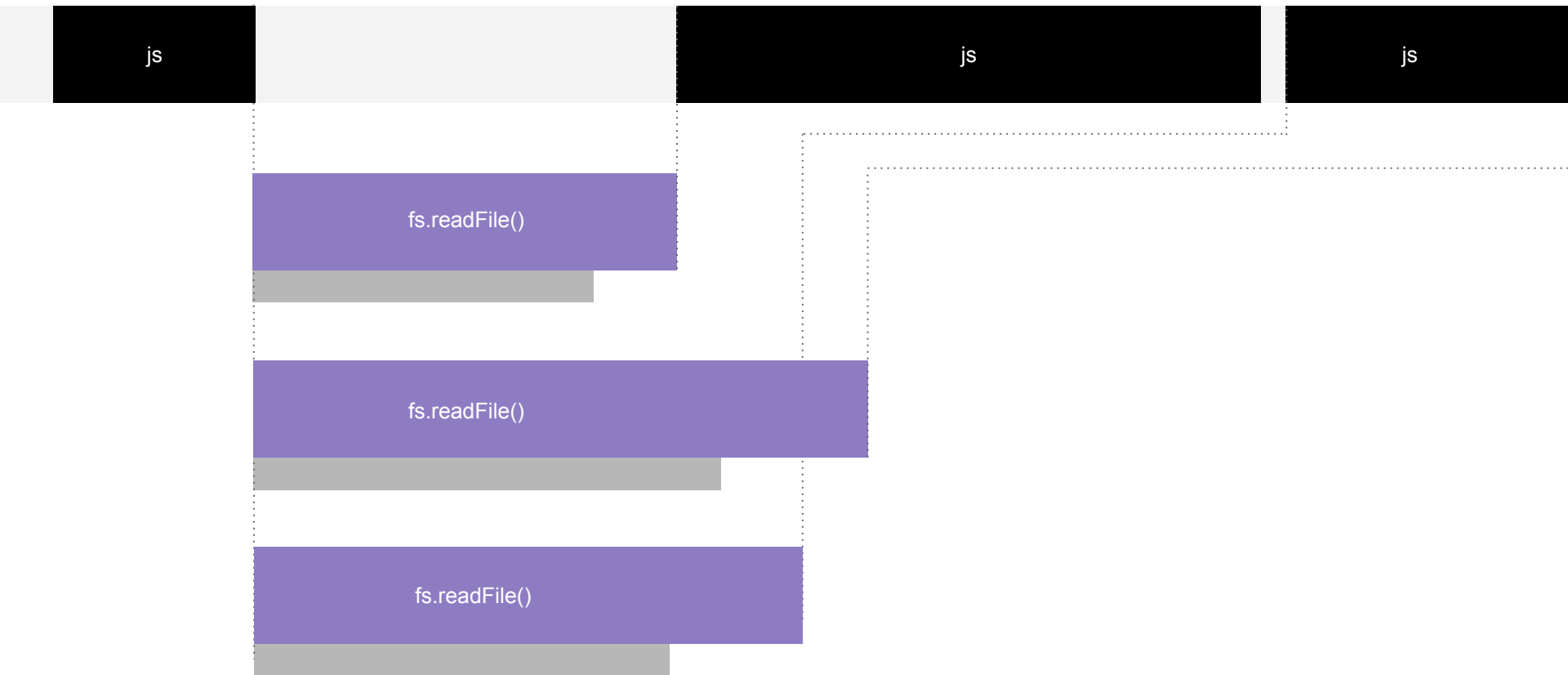
js

fs.readFile()

js

js

fs.readFile()



# Streams

- ¿Qué hacemos para procesar un fichero de 200gb?

# Streams

- ¿Qué hacemos para procesar un fichero de 200gb?
  - ¡Lo procesamos por partes!
  - Leemos un fragmento y lo procesamos...
  - Una y otra vez
  - Hasta que hayamos procesado el fichero entero

# Streams

- **Streams**
  - **Observables** que representan un **flujo** de datos
  - Cuatro tipos:
    - lectura
    - escritura
    - dúplex
    - transformación

# Streams

- **Streams**
  - Son **complejos**
    - mecanismo fundamental en node
    - realizan una tarea delicada
  - Interfaces diferentes de **lectura** y **escritura**

# Streams de Lectura

- Un flujo de datos “entrantes”
- Procesarlos poco a poco
- Mediante eventos:
  - data
  - end
  - error

```
const fs = require('fs')

const fileStream = fs.createReadStream(__filename)

fileStream.on('data', (data) => {
  console.log('read:', data)
})

fileStream.on('end', () => {
  console.log('done!')
})

fileStream.on('error', (err) => {
  console.error(err)
})
```



```
const fs = require('fs')
```

```
const fileStream = fs.createReadStream(__filename)
```

```
fileStream.on('data', (data) => {  
  console.log('read:', data)  
})
```

```
fileStream.on('end', () => {  
  console.log('done!')  
})
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
})
```

```
const fs = require('fs')
```

```
const fileStream = fs.createReadStream(__filename)
```

```
fileStream.on('data', (data) => {  
  console.log('read:', data)  
})
```

```
fileStream.on('end', () => {  
  console.log('done!')  
})
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
})
```

```
$ node test.js
```

```
read: <Buffer 63 6f 6e 73 74 20 66 73 20 3d 20 72 65  
71 75 69 72 65 28 27 66 73 27 29 0a 0a 63 6f 6e 73  
74 20 66 69 6c 65 53 74 72 65 61 6d 20 3d 20 66 73  
2e 63 72 ... >  
done!
```

```
$ node test.js
```

```
read: <Buffer 63 6f 6e 73 74 20 66 73 20 3d 20 72 65  
71 75 69 72 65 28 27 66 73 27 29 0a 0a 63 6f 6e 73  
74 20 66 69 6c 65 53 74 72 65 61 6d 20 3d 20 66 73  
2e 63 72 ... >
```

```
done!
```

```
const fs = require('fs')

const fileStream = fs.createReadStream(__filename)

fileStream.on('data', (data) => {
  console.log('read:', data.toString())
})

fileStream.on('end', () => {
  console.log('done!')
})

fileStream.on('error', (err) => {
  console.error(err)
})
```

# Streams de Escritura

- Un flujo de datos “salientes”
- Escribir data poco a poco
- Dos métodos esenciales:
  - `write(chunk, [encoding, callback])`
  - `end([chunk, encoding, callback])`

```
const fs = require('fs')

const fileStream = fs.createReadStream(__filename)
const outputStream = fs.createWriteStream('/tmp/copy.out')

fileStream.on('data', (data) => {
  outputStream.write(data)
})

fileStream.on('end', () => {
  outputStream.end()
})

fileStream.on('error', (err) => {
  console.error(err)
})
```

```
const fs = require('fs')
```

```
const fileStream = fs.createReadStream(__filename)
```

```
const outputStream = fs.createWriteStream('/tmp/copy.out')
```

```
fileStream.on('data', (data) => {  
  outputStream.write(data)  
}))
```

```
fileStream.on('end', () => {  
  outputStream.end()  
}))
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
}))
```



```
const fs = require('fs')
```

```
const fileStream = fs.createReadStream(__filename)
```

```
const outputStream = fs.createWriteStream('/tmp/copy.out')
```

```
fileStream.on('data', (data) => {  
  outputStream.write(data)  
})
```

```
fileStream.on('end', () => {  
  outputStream.end()  
})
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
})
```

```
const fs = require('fs')

const fileStream = fs.createReadStream(__filename)
const outputStream = fs.createWriteStream('/tmp/copy.out')

fileStream.pipe(outputStream)
```

# Ejercicio

- reescribe el clon de grep...
  - utilizando **streams**
  - para que funcione con ficheros de cualquier tamaño

**http**

# Http

- `http.createServer(requestListener)`
  - crea una instancia de `http.Server`
  - *requestListener* se invoca **en cada petición** con dos parámetros:
    - **req**: stream de lectura con info sobre la petición
    - **res**: stream de escritura para enviar la respuesta

```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
```

```
const server = http.createServer((req, res) => {  
  console.log('> conexión!')  
  res.write('Hola, Mundo!')  
  res.end()  
})
```

```
server.listen(3000)  
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
```

```
const server = http.createServer((req, res) => {  
  console.log('> conexión!')  
  res.write('Hola, Mundo!')  
  res.end()  
})
```

```
server.listen(3000)  
console.log('* Ready http://localhost:3000')
```



```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})
```

```
server.listen(3000)
console.log('* Ready http://localhost:3000')
```

# Ejercicio

- Escribe un **programa** que sirva **su propio código** por **HTTP**
  - <http://localhost:3000/>
  - ruta del script en **\_\_filename**

# Http

- Para testear un `httpServer` vamos a usar:
  - `supertest`
  - <https://github.com/visionmedia/supertest>
  - Monta la instancia en un puerto temporal, hace la petición y limpia al terminar
  - Matchers específicos para peticiones HTTP

```
const request = require('supertest')

describe('GET /user', () => {
  it('respond with json', (done) => {
    const response = request(app)
      .get('/user')
      .set('Accept', 'application/json')
    response
      .expect('Content-Type', /json/)
      .expect(200)
      .end((err) => {
        if (err) return done(err)
        done()
      })
  })
})
```

```
const request = require('supertest')

describe('GET /user', () => {
  it('respond with json', (done) => {
    const response = request(app)
      .get('/user')
      .set('Accept', 'application/json')
    response
      .expect('Content-Type', /json/)
      .expect(200)
      .end((err) => {
        if (err) return done(err)
        done()
      })
  })
})
```

```
const request = require('supertest')

describe('GET /user', () => {
  it('respond with json', (done) => {
    const response = request(app)
      .get('/user')
      .set('Accept', 'application/json')
    response
      .expect('Content-Type', /json/)
      .expect(200)
      .end((err) => {
        if (err) return done(err)
        done()
      })
  })
})
```



# Http

- `req.url`
  - dirección solicitada por el cliente
- `req.headers`
  - cabeceras proporcionadas por el cliente
- `req` leído como *stream*
  - cuerpo de la petición (POST)

# Http

- `res.writeHead(statusCode, headerObject)`
  - status code y las cabeceras de la respuesta
- `res.statusCode`
  - asignar un número para especificar un status code *sin especificar ninguna cabecera*
- `res` escrito como *stream*
  - cuerpo de la respuesta

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')
```

```
const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})
```

```
server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

# Ejercicio

- Escribe una **aplicación web** para “acortar” enlaces con tres “rutas”:
  - un formulario para introducir el target
  - una página de confirmación con el link creado
  - una ruta de redirección que recibe el token como parámetro



# mysql

# MySQL

- Node.js **no** trae soporte para bases de datos...
  - ni para ningún otro servicio que no sea general del SO
- Interfaz C++ para integrar **librerías externas**
  - extender la plataforma
  - transparente para nosotros
  - simplemente instalamos el módulo adecuado

# MySQL

- Los paquetes que integran librerías externas...
  - no siempre siguen la misma filosofía de diseño
  - no siempre siguen el interfaz `callback(err, data...)`
  - no siempre son estables o están bien mantenidos
  - a veces requieren un **compilador de c++**
  - no siempre están bien **documentados**

# MySQL

- El paquete **mysql**
  - 270k descargas semanales
  - 12k estrellas en github
  - múltiples commits en el último mes
  - documentación adecuada
  - interfaz bastante fiel al estilo de Node.js

```
$ npm install mysql
```

```
CREATE DATABASE test;  
USE test;
```

```
CREATE TABLE users (  
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(256),  
    email VARCHAR(256),  
    password VARCHAR(256)  
);
```

```
INSERT INTO USERS VALUES (  
    null, 'Elias', 'eliasagc@gmail.com', 'supersecret'  
);
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})

connection.connect();

connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})

connection.end()
```

```
const mysql = require('mysql')
```

```
const connection = mysql.createConnection({  
  host: 'localhost',  
  user: 'root',  
  database: 'test'  
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users', (err, results, fields) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(results)  
})
```

```
connection.end()
```



```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users' (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})

connection.connect();

connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})

connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})

connection.connect();

connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})

connection.end()
```

```
connection.query(  
  'INSERT INTO users VALUES (null, "Homer", "h.simpson@fox.com", "rosquilla")',  
  (err, result) => console.log('Inserted row ID:', result.insertId)  
)
```

```
connection.query(  
  'UPDATE users SET email = "homer@sprngfld.usa" WHERE name = "Homer"',  
  (err, result) => console.log('Changed rows:', result.changedRows)  
)
```

```
connection.query(  
  'DELETE FROM users WHERE name = "Homer"',  
  (err, result) => console.log('Affected rows:', result.affectedRows)  
)
```

```
connection.query(  
  'INSERT INTO users VALUES (null, "Homer", "h.simpson@fox.com", "rosquilla")',  
  (err, result) => console.log('Inserted row ID:', result.insertId)  
)
```

```
connection.query(  
  'UPDATE users SET email = "homer@sprngfld.usa" WHERE name = "Homer"',  
  (err, result) => console.log('Changed rows:', result.changedRows)  
)
```

```
connection.query(  
  'DELETE FROM users WHERE name = "Homer"',  
  (err, result) => console.log('Affected rows:', result.affectedRows)  
)
```

# MySQL

- Seguridad básica: escapa todo lo que venga del usuario!
  - **mysql.escape(*anyValue*)**
  - placeholders en **mysql.query(...)**



```
connection.query(  
    'SELECT * FROM users WHERE email = ?',  
    ['eliasagc@gmail.com'],  
    (err, results, fields) => {  
        err ? console.error(err) : console.log(results)  
    })  
)
```

```
connection.query(  
  'SELECT * FROM users WHERE email = ?',  
  ['eliasagc@gmail.com'],  
  (err, results, fields) => {  
    err ? console.error(err) : console.log(results)  
  })  
)
```

```
const data = {  
  name: 'Homer',  
  email: 'h.simpson@fox.com',  
  password: 'rosquilla'  
}
```

```
connection.query('INSERT INTO users SET ?', data)
```

# MySQL

- Para trabajar con gran volumen de datos...
  - podemos convertir el resultado en un *stream*
  - que se trae los resultados poco a poco
  - para poder procesarlos según llegan

```
const readStream = connection
    .query('SELECT * FROM users')
    .stream()

readStream.on('data', row => console.log('row:', row))
readStream.on('end', () => console.log('done!'))
```

# MySQL

- La librería tiene funcionalidad más avanzada
  - pool de conexiones
  - transacciones
  - procedimientos almacenados
  - gestión de usuarios
  - etc....

# Ejercicio

- Modifica el acortador de urls para que guarde los enlaces en **mysql**
  - y que cuente cuantas veces se visita cada enlace