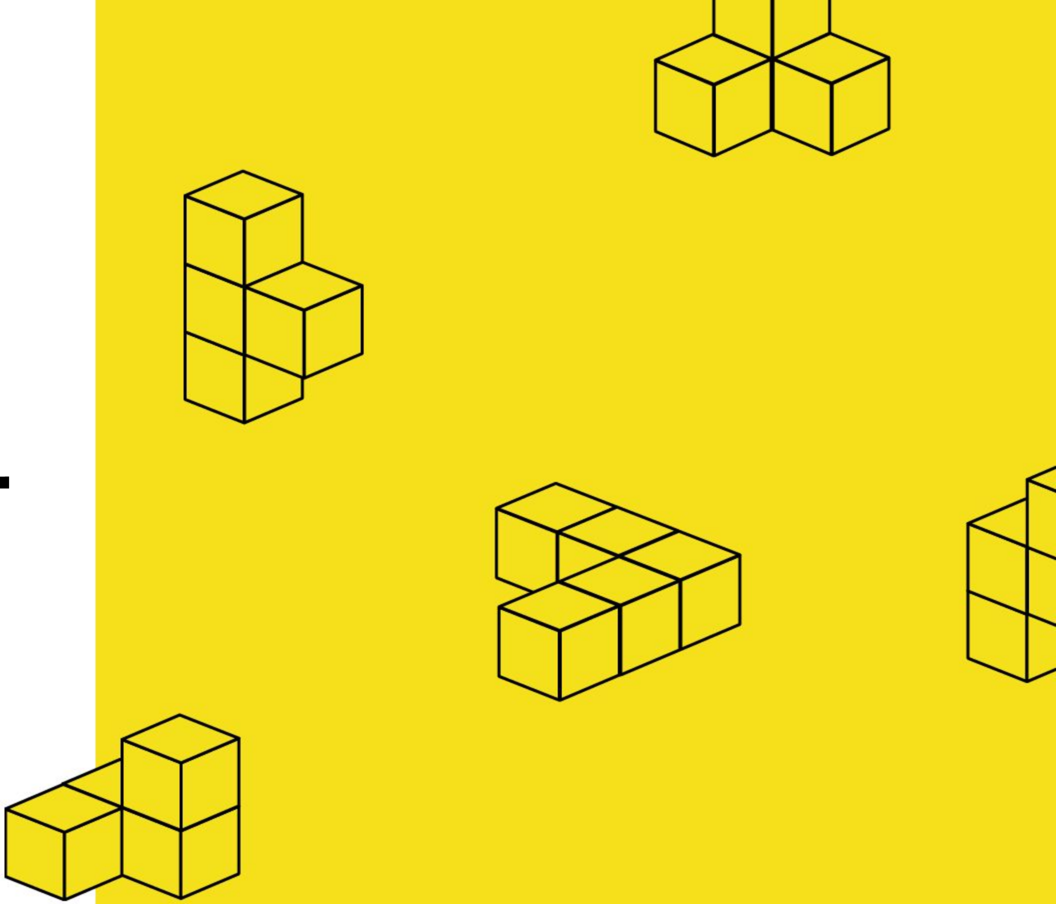
Escuela de JavaScript



Escuela de JavaScript | Platzi

Agenda de Actividades:

- 09:15 - 09:30 Recapitulación Curso de Backend con Node
- 09:30 - 09:45 Q&A
- 09:45 - 10:15 Clase de Práctica
- 10:15 - 10:45 Resolver Retos
- 10:45 - 12:20 Mentoría sobre proyectos

Código de Conducta

Recapitulación

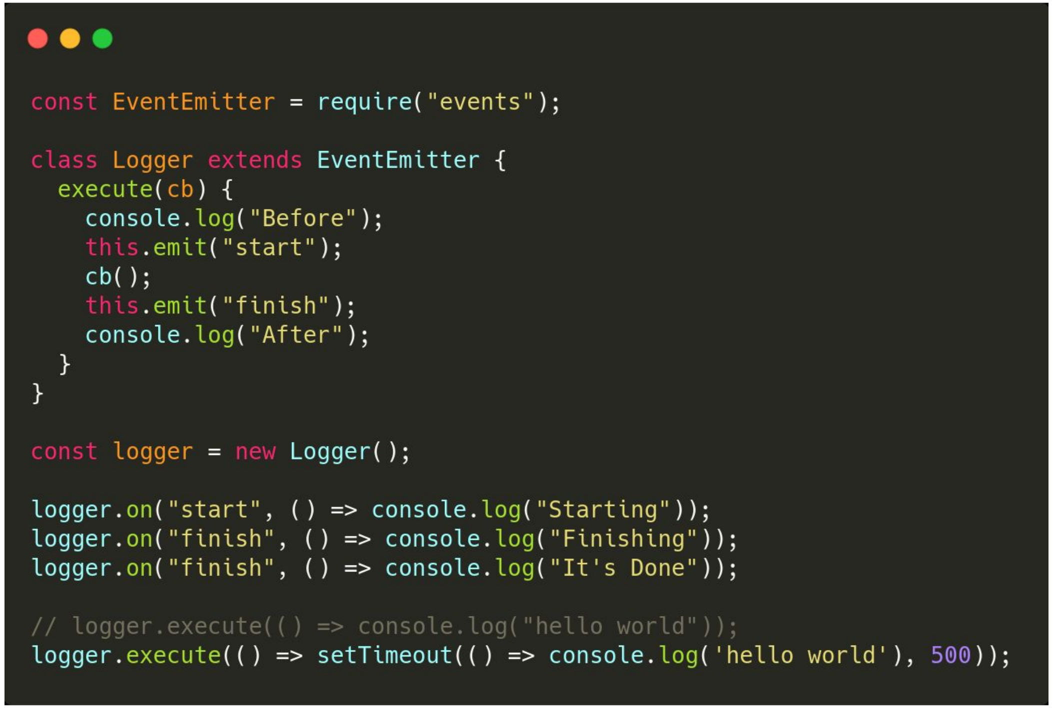
Curso de Backend con Node

Escuela de JavaScript | Platzi

¿Qué es Node.js?

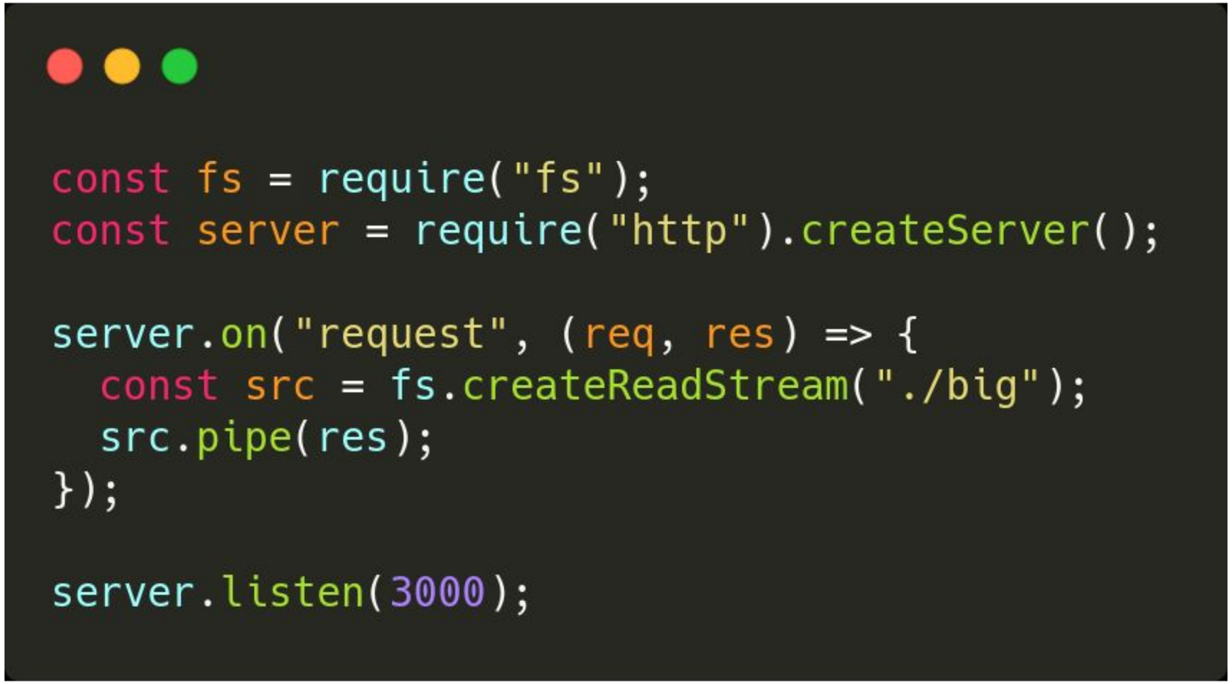
Diferencias entre Node.js y JavaScript

Arquitectura orientada a eventos



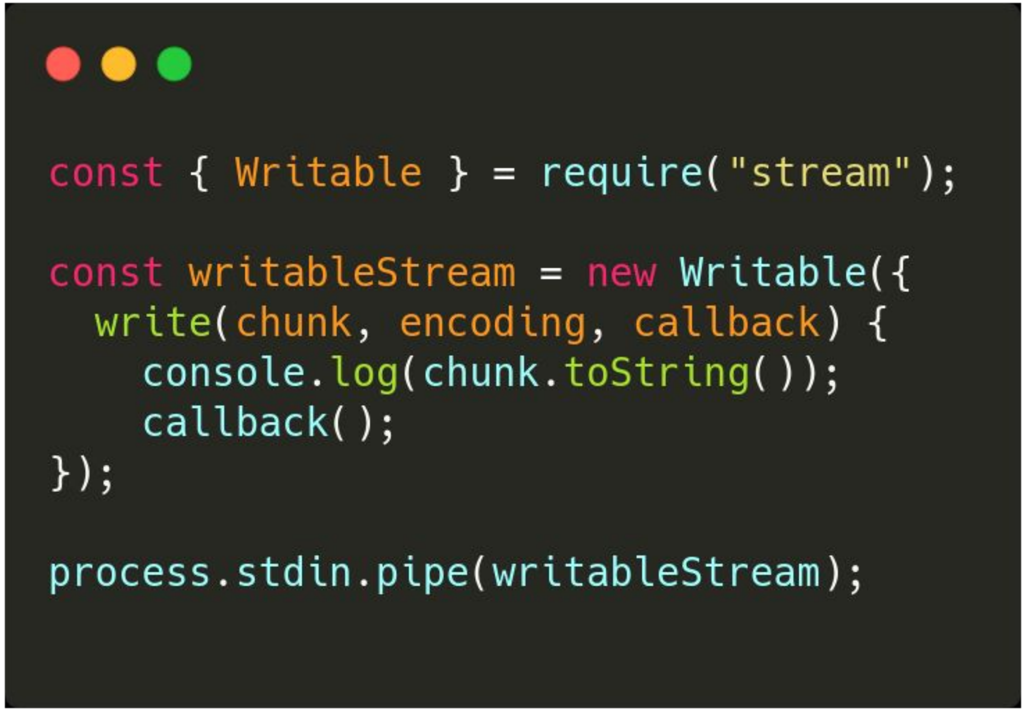
```javascript
const EventEmitter = require("events");

class Logger extends EventEmitter {
  execute(cb) {
    console.log("Before");
    this.emit("start");
    cb();
    this.emit("finish");
    console.log("After");
  }
}

const logger = new Logger();

logger.on("start", () => console.log("Starting"));
logger.on("finish", () => console.log("Finishing"));
logger.on("finish", () => console.log("It's Done"));

// logger.execute(() => console.log("hello world"));
logger.execute(() => setTimeout(() => console.log('hello world'), 500));
```

Streams



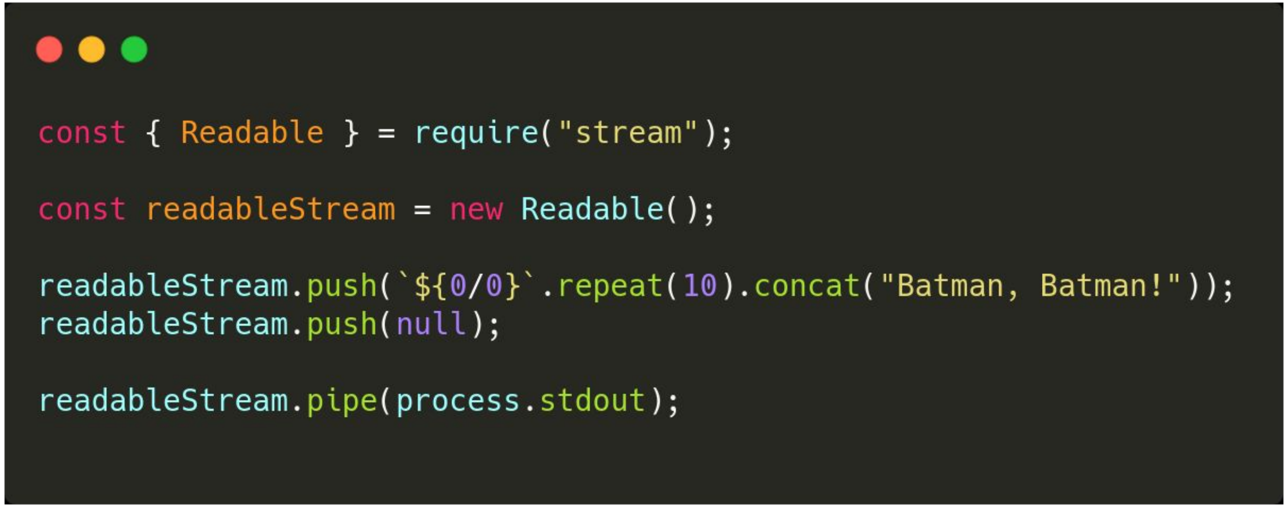
```
const fs = require("fs");
const server = require("http").createServer();

server.on("request", (req, res) => {
  const src = fs.createReadStream("./big");
  src.pipe(res);
});

server.listen(3000);
```

Writable Streams



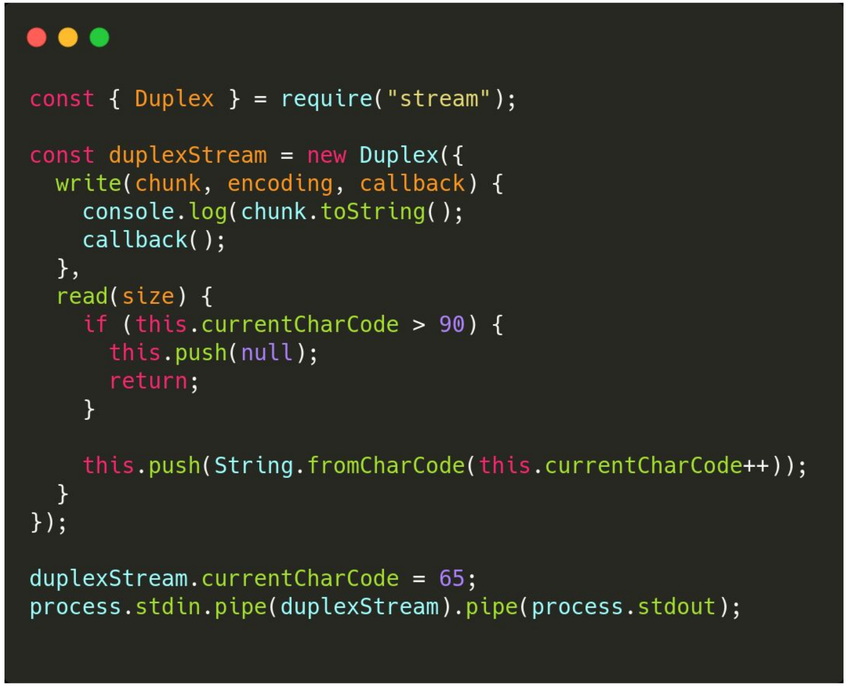
```javascript
const { Writable } = require("stream");

const writableStream = new Writable({
  write(chunk, encoding, callback) {
    console.log(chunk.toString());
    callback();
});

process.stdin.pipe(writableStream);
```

Readable Streams

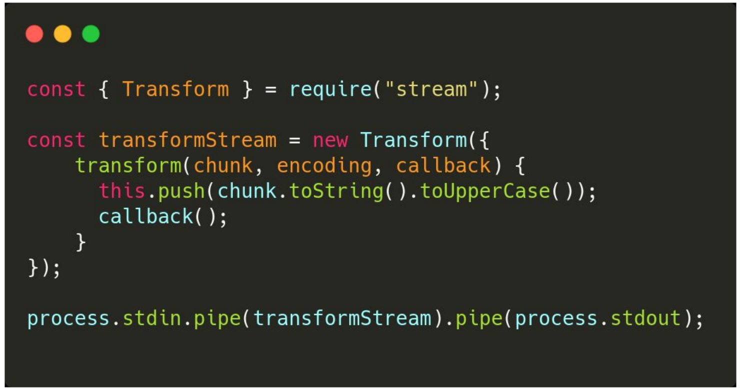


```
const { Readable } = require("stream");

const readableStream = new Readable();

readableStream.push(`${0/0}`.repeat(10).concat("Batman, Batman!"));
readableStream.push(null);

readableStream.pipe(process.stdout);
```

Duplex y Transforms Streams

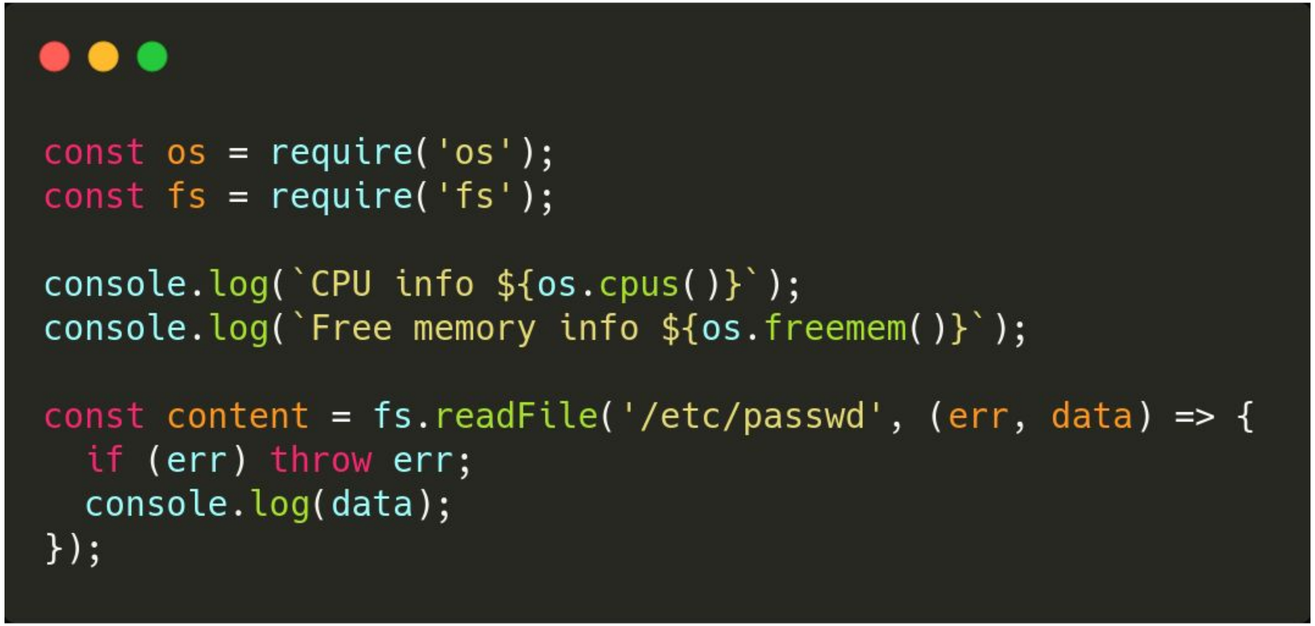


```javascript
const { Duplex } = require("stream");

const duplexStream = new Duplex({
  write(chunk, encoding, callback) {
    console.log(chunk.toString());
    callback();
  },
  read(size) {
    if (this.currentCharCode > 90) {
      this.push(null);
      return;
    }

    this.push(String.fromCharCode(this.currentCharCode++));
  }
});

duplexStream.currentCharCode = 65;
process.stdin.pipe(duplexStream).pipe(process.stdout);
```



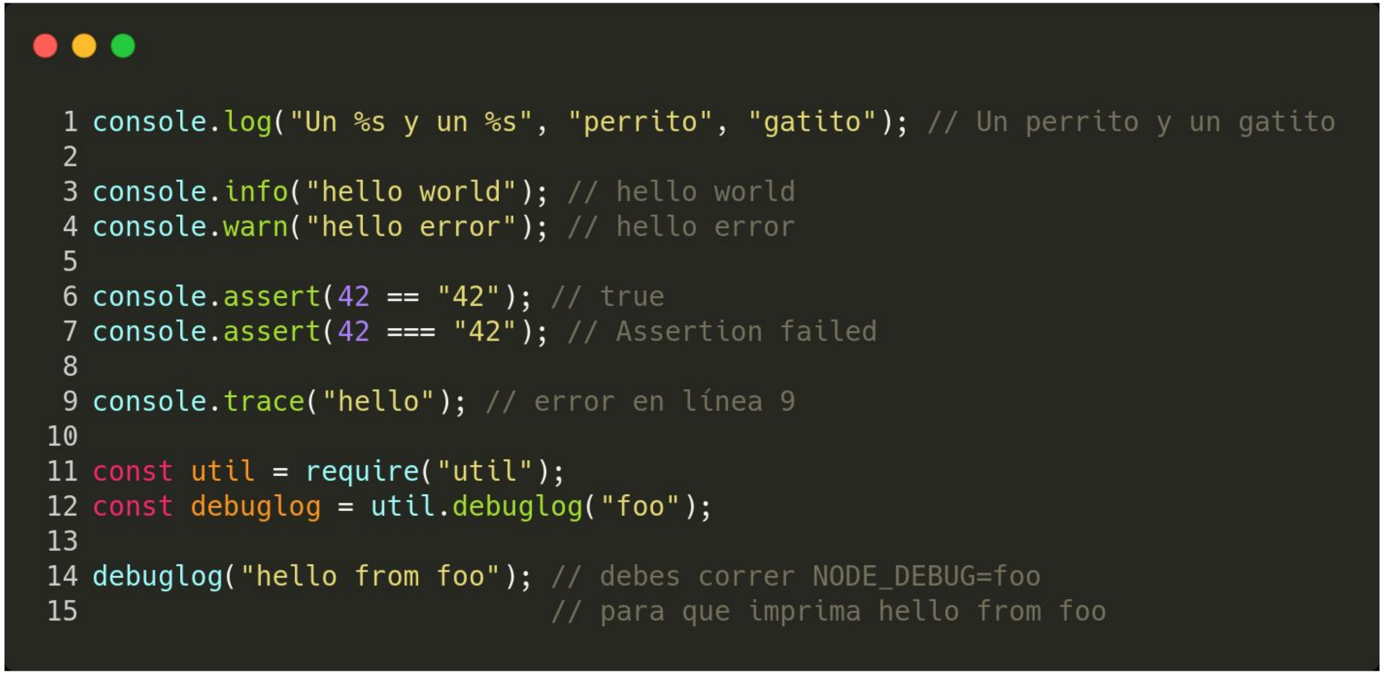
```javascript
const { Transform } = require("stream");

const transformStream = new Transform({
    transform(chunk, encoding, callback) {
      this.push(chunk.toString().toUpperCase());
      callback();
    }
});

process.stdin.pipe(transformStream).pipe(process.stdout);
```

Módulos os y fs



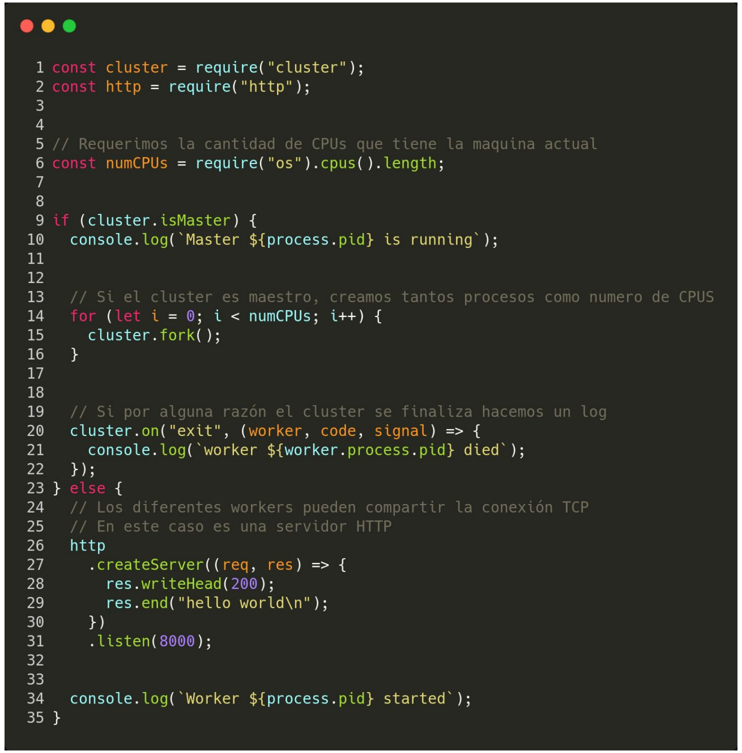
```
const os = require('os');
const fs = require('fs');

console.log(`CPU info ${os.cpus()}`);
console.log(`Free memory info ${os.freemem()}`);

const content = fs.readFile('/etc/passwd', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

console y util



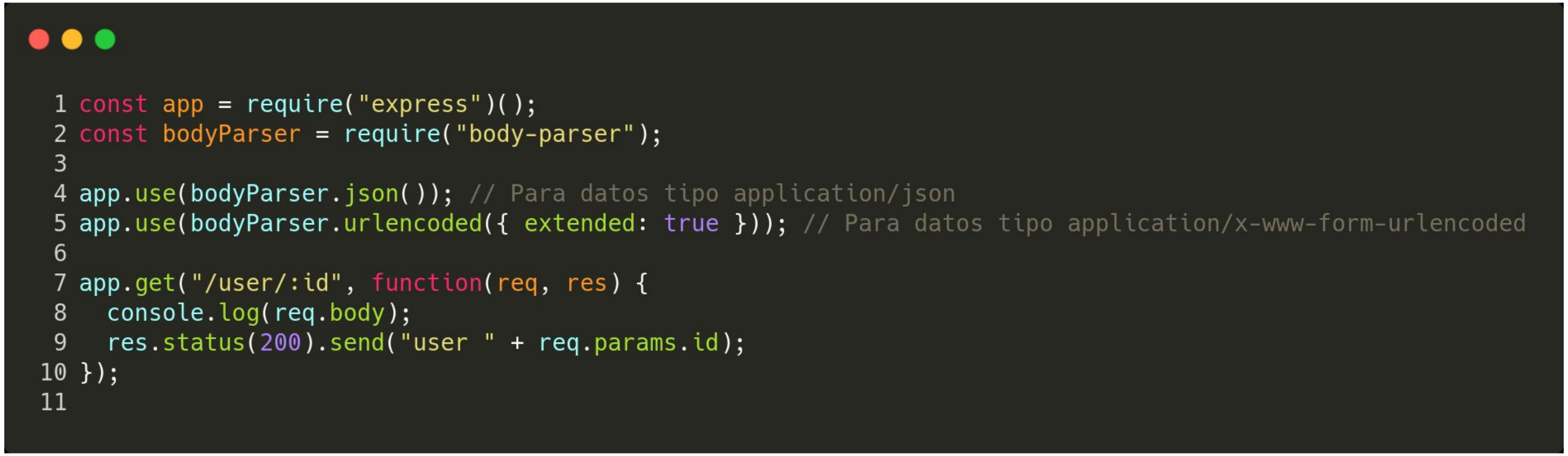
```
 1 console.log("Un %s y un %s", "perrito", "gatito"); // Un perrito y un gatito
 2
 3 console.info("hello world"); // hello world
 4 console.warn("hello error"); // hello error
 5
 6 console.assert(42 == "42"); // true
 7 console.assert(42 === "42"); // Assertion failed
 8
 9 console.trace("hello"); // error en línea 9
10
11 const util = require("util");
12 const debuglog = util.debuglog("foo");
13
14 debuglog("hello from foo"); // debes correr NODE_DEBUG=foo
15                             // para que imprima hello from foo
```

Clusters



```javascript
const cluster = require("cluster");
const http = require("http");


// Requerimos la cantidad de CPUs que tiene la maquina actual
const numCPUs = require("os").cpus().length;


if (cluster.isMaster) {
  console.log(`Master ${process.pid} is running`);


  // Si el cluster es maestro, creamos tantos procesos como numero de CPUS
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }


  // Si por alguna razón el cluster se finaliza hacemos un log
  cluster.on("exit", (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
} else {
  // Los diferentes workers pueden compartir la conexión TCP
  // En este caso es una servidor HTTP
  http
    .createServer((req, res) => {
      res.writeHead(200);
      res.end("hello world\n");
    })
    .listen(8000);


  console.log(`Worker ${process.pid} started`);
}
```

Express: request y response



```javascript
const app = require("express")();
const bodyParser = require("body-parser");

app.use(bodyParser.json()); // Para datos tipo application/json
app.use(bodyParser.urlencoded({ extended: true })); // Para datos tipo application/x-www-form-urlencoded

app.get("/user/:id", function(req, res) {
  console.log(req.body);
  res.status(200).send("user " + req.params.id);
});

```

Parsers

Query strings para métodos GET
http://localhost:3000?field1=value&field2=value2

Para métodos POST, los key-value pairs vienen en el request body
x-www-form-urlencoded

Cuando subes archivos, debes partir los datos en pedazos (‘chunks’)
multipart/form-data

Cuando mandas un JSON, el body es de tipo
raw/json

Para qué express sea capaz de agarrar los datos del request, necesita de un poco de ayuda
Middlewares (body parser, multer, etc.)

¡Vamos al código!

¿Dudas?

Escuela de JavaScript | Platzi

Reto:

Repo https://github.com/platzi/escuelajs-reto-08

Escuela de
JavaScript