Javascript avanzado (Incompleto)

Errores

Errores

En Javascript se pueden dar diferentes tipos de errores

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

Errores

Podemos lanzar errores manualmente utilizando **throw**

```
throw EvalError
```

Se parará la ejecución del código y se mostrará el error en consola

Errores

Podemos crear errores personalizados

```
throw value       // String, Number, Boolean or Object
```

Se parará la ejecución del código y se mostrará el error en consola

Errores

Podemos controlar errores utilizando **try / catch**

```
try{
      console.olg("Lorem Ipsum")
}
```

El código que se ejecuta en el body de un **try** no genera errores

Errores

Dejar que los errores sucedan **silenciosamente** no es recomendable

```
try{
      console.olg("Lorem Ipsum")
} catch (err){
      console.log(err.name)
      console.log(err.message)
}
```

catch nos permite ejecutar código en caso de error

Además nos da acceso al objeto de Error, que contiene información sobre el error

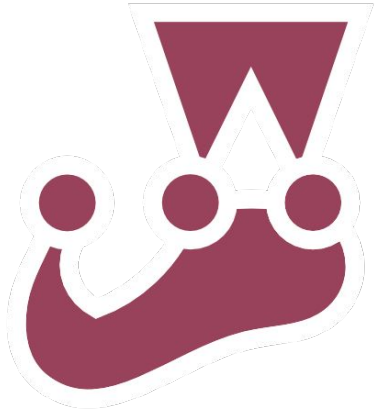
Errores

Aunque generemos un error, podemos ejecutar código en **finally**

```
try{
      console.olg("Lorem Ipsum")
} catch (err){
      console.log(err.name)
      console.log(err.message)
} finally() {
      // este código se ejecuta pase lo que pase
}
```

Incluso si el error sucede dentro de catch

Demo try/catch



Jest

Jest

Jest es un **testing framework** de Javascript muy sencillo de utilizar

Los tests nos permiten comprobar que diferentes piezas de nuestro código devuelven los resultados esperados

Cuando realizamos un cambio en la aplicación podemos correr todos los tests y comprobar que ese cambio no ha roto ninguna de las piezas

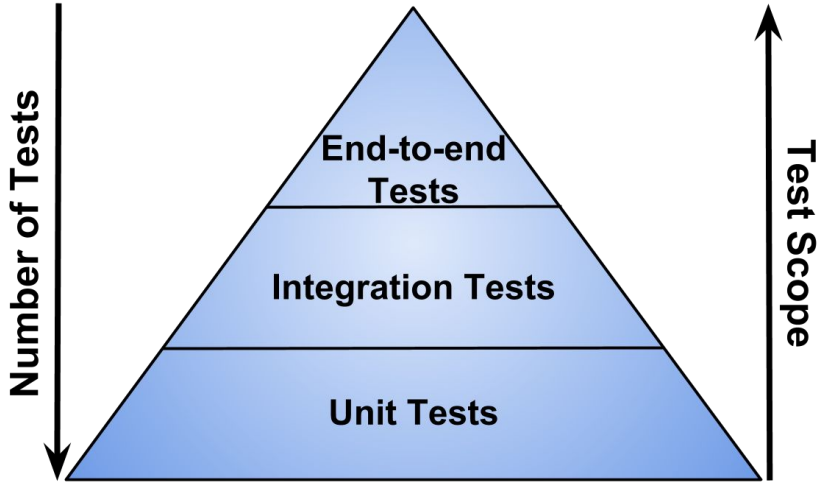
Testing

Los tests son una parte fundamental del desarrollo software

Algunos equipos de desarrollo aplican **test-driven development**

Testing

Hay diferentes niveles de test en función de su alcance



Testing

La mayoría de lenguajes relevantes tienen testing frameworks como Jest para realizar tests unitarios y tests de integración

Los tests end-to-end a menudo requieren herramientas más avanzadas que permitan simular las acciones del usuario final (eg: Selenium)

Jest

npm permite instalar Jest fácilmente

```
npm install jest --save-dev
```

Para poder lanzar los tests desde npm es necesario añadir un script en **package.json**:

```
{
  "scripts": {
    "test": "jest"
  }
}
```

Jest

```
// Fichero en nuestro código
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

Los tests se crean en ficheros **nombre.test.js (**este formato permite a Jest encontrarlos)

```
// sum.test.js
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Jest

Lanzamos los tests con npm utilizando el script creado

```
npm run test
```

El comando test nos muestra el resultado de todos los tests

```
PASS  ./sum.test.js
✓ adds 1 + 2 to equal 3 (5ms)
```

Matchers

Matchers

Expect crea un **expectation object** al que podemos aplicar **matchers**

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4)
});
```

El matcher **.toBe()** compara la igualdad exacta entre dos valores (**===**)

Matchers

Si queremos comparar el valor de dos objetos no nos vale con igualdad exacta. **toEqual()** efectua **deepComparison**

```
test('object assignment', () => {
  const data = {one: 1}
  data['two'] = 2
  expect(data).toEqual({one: 1, two: 2})
});
```

Matchers

También es posible comprobar la no igualdad

```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0)
    }
  }
})
```

Matchers

Herramientas para gestionar truthiness

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});
```

Matchers

Comparación numérica

```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);
  expect(value).toBeLessThan(5);
  expect(value).toBeLessThanOrEqual(4.5);
});
```

Matchers

Expresiones regulares

```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});

test('but there is a "stop" in Christoph', () => {
  expect('Christoph').toMatch(/stop/);
});
```

Matchers

Arrays

```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it', () => {
  expect(shoppingList).toContain('beer');
});
```

Ejercicio Jest

- Ruta: https://github.com/antonio-redradix/jest-exercises
- Crea tests para las funciones del fichero utils.js

Ejercicio Jest II

- Ruta: https://github.com/antonio-redradix/jest-exercises
- Crea tests para las funciones del fichero parser.js

Setup and Teardown

Setup and Teardown

En ocasiones debemos realizar las mismas tareas una y otra vez antes de cada test

- Eg: Inicializar una base de datos

```
beforeEach(() => {
  initializeCityDatabase() ;
});

afterEach(() => {
  clearCityDatabase() ;
});

test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy() ;
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy() ;
});
```

Setup and Teardown

En ocasiones debemos realizar una tarea **antes de poder ejecutar cualquier test**

```
beforeAll(() => {
  return initializeCityDatabase();
});

afterAll(() => {
  return clearCityDatabase();
});

test('city database has Vienna' , () => {
  expect(isCity( 'Vienna')).toBeTruthy();
});

test('city database has San Juan' , () => {
  expect(isCity( 'San Juan')).toBeTruthy();
});
```

Setup and Teardown

En ocasiones debemos realizar una tarea **antes de poder ejecutar cualquier test**

```
beforeAll(() => {
  return initializeCityDatabase();
});

afterAll(() => {
  return clearCityDatabase();
});
```

Si beforeAll y afterAll devuelven promesas los tests no se ejectuaran hasta que estas se cumplan

Ejercicio Jest II

- Ruta: https://github.com/antonio-redradix/jest-exercises
- Crea tests para las funciones del fichero db.js

Typescript

Typescript

Lenguaje de programación desarrollado por Microsoft

Es un **superset** de Javascript

Está diseñado para **transpilarse** a Javascript

Webpack

npm permite instalar typescript fácilmente

```
npm install typescript --save
```

En este repositorio hay una configuración de **webpack** preparada para proyectos Typescript

Explicación configuración

Typescript

Javascript es un lenguaje liberal con **dynamic typing**

En aplicaciones grandes, la libertad que ofrece Javascript puede acabar generando bases de código caóticas e inestables

Typescript permite utilizar **static typing** en **Javascript.** Forzando a los desarrolladores a someterse a reglas que facilitan el orden

Typescript

Históricamente, Typescript también añadía features que no estaban incluídas en **ES5**

- let / const
- clases
- arrow functions
- Etc.

Type annotations

Type annotations

Fuerzan el **tipo** de una variable

```
let text: string = "Lorem ipsum"


function print(str: string){
    console.log(str)
}

class Student {
    fullName: string
}
```

Si asignamos un valor del tipo incorrecto provocaremos un **error**

Type annotations

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

Forzando que los parámetros de una función tengan un tipo específico **evitamos futuros errores**

Type annotations

```
function greet(person: string){
    console.log(`Hello, ${person}!`)
}
```

Las anotaciones también nos ayudan a entender el **objetivo** de un parámetro

Type annotations

También podemos forzar **el tipo de una función** (del valor que devuelve)

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

Si no devuelve valor utilizamos **void**

```
function warnUser(): void {
    console.log("This is a warning message");
}
```

Type annotations

Listado de tipos básicos

```
number
string
boolean
object
undefined
null
void
any         // útil si no sabemos el tipo de antemano
```

Type annotations

Existen dos formas de declarar type annotations en arrays

```
let numbers: number[] = [1, 2, 3]
let letters: string[] = ["a","b","c"]
```

```
let numbers: Array<number> = [1, 2, 3]
let letters: Array<string> = ["a","b","c"]
```

Interfaces

Interfaces

Las **interfaces** especifican los tipos que ha de tener un objeto

```
interface Person {
    firstName: string;
    lastName: string;
}
```

Las **interfaces** no asignan valores!

Interfaces

Podemos utilizar las **interfaces** como un **tipo** cualquiera

```
interface Person {
    firstName: string;
    lastName: string;
}

let person: Person
person = {firstName: "David", lastName: "Lynch"}
```

Si person no tiene los campos y tipos adecuados provocaremos un **error**

Interfaces

Las interfaces son muy útiles para forzar consistencia en nuestro código

```
interface Coords {
    lat: number;
    lon: number;
}
```

```
function search(coords: Coords) : Place {
    //...
}
```

Interfaces

Si asignamos la **interfaz** en toda nuestra base de código **evitamos variaciones**:

```
let coords = [40.81, -3.99]
let coords = [-3.99, 40.81]
let coords = ["40.81", "-3.99"]
let coords = {latitude: 40.81, longitude: -3.99}
let coords = {lat: 40.81, lon: -3.99}
let coords = {lat: 40.81, long: -3.99}
let coords = {lat: "40.81", lon: "-3.99"}
//…

search(coords)
```

Interfaces

Las interfaces ayudan a la **legibilidad**

```
interface Place {
    name: string;
    address: string;
}
```

```
function search(coords: Coords) : Place {
    //...
}
```

Ejercicio Typescript

- Ruta: https://github.com/antonio-redradix/jest-exercises
- Convierte los ficheros a ficheros .ts
- Anota el código JS con type annotations

APIs Javascript

Web Storage

Web Storage

Cada vez que cargamos (o recargarmos) una página se ejecuta su código Javascript y se inicializan sus variables

Los navegadores modernos nos ofrecen mecanismos para guardar información para futuras visitas a la misma página

Uno de esos mecanismos es la **web storage API**

Web Storage

Web storage se compone de dos mecanismos

- **sessionStorage**: mantiene la información hasta que se cierra el navegador (recargar la página mantiene la información)

- **localStorage**: mantiene la información incluso cuando se cierra el navegador y se vuelve a abrir

Web Storage

Estos mecanismos son accesibles a partir de las propiedades

```
window.sessionStorage
```

```
window.localStorage
```

Estas propiedades son **objetos** a los que podemos asignar valores

```
sessionStorage.numberOfVisits = 0
```

```
sessionStorage.numberOfVisits = parseInt(sessionStorage.numberOfVisits) + 1
```

Por compatibilidad con antiguos navegadores los valores se guardan **en forma de string**

Demo

Ejemplos:
sessionStorage
localStorage

Web Storage

Eliminar un conjunto clave/valor

```
localStorage.removeItem(key);
```

```
localStorage.removeItem("numberOfVisits");
```

Ejercicio webStorage

Crea una página que muestre el número de veces que ha sido visitada y un botón

Si se pulsa el botón el contador de visitas debe reiniciarse