



Repaso de JavaScript



Repaso Javascript

Introducción:

Es un lenguaje ligero, interpretado y orientado a objetos con funciones de primera clase

JavaScript fue inventado por Brendan Eich en 1995 y se convirtió en un estándar ECMA en 1997

Las versiones de ECMAScript se han abreviado como ES1, ES2, ES3, ES5 y ES6

Desde 2016, las nuevas versiones se nombran por año (ECMAScript 2016/2017/2018).

La implementación de ECMAScript la gestiona el grupo de trabajo TC39 <https://tc39.es/>

Repaso Javascript

Introducción:

ES5	ECMAScript 5 (2009) Read More	Added "strict mode" Added JSON support Added String.trim() Added Array.isArray() Added Array iteration methods Allows trailing commas for object literals
ES6	ECMAScript 2015 Read More	Added let and const Added default parameter values Added Array.find() Added Array.findIndex()
	ECMAScript 2016 Read More	Added exponential operator (**) Added Array.includes()
	ECMAScript 2017 Read More	Added string padding Added Object.entries() Added Object.values() Added async functions Added shared memory
	ECMAScript 2018 Read More	Added rest / spread properties Added asynchronous iteration Added Promise.finally() Additions to RegExp

Soporte de JavaScript

[illegible][illegible]

Repaso Javascript

Babel.js

Transforma código escrito con sintaxis nueva de JavaScript en su sintaxis equivalente para versiones anteriores.

```
6     "scripts": {
7       "build": "babel src --out-dir build",
8       "prod": "node .",
9       "start": "babel-node src/main.js",
0       "dev": "nodemon --exec babel-node -- src/main.js",
1       "test": "echo \"Error: no test specified\" && exit 1"
2     },
3     "author": "Juanjofp",
4     "license": "ISC",
5     "devDependencies": {
6       "@babel/cli": "^7.14.8",
7       "@babel/core": "^7.15.0",
8       "@babel/node": "^7.14.9",
9       "@babel/preset-env": "^7.15.0",
0       "eslint": "^7.32.0",
1       "nodemon": "^2.0.12",
2       "prettier": "^2.3.2"
3     },
4     "babel": {
5       "presets": [
6         "@babel/preset-env"
7       ]
8     }
9   }
}
```



Repaso Javascript

Ejercicios

Crear un proyecto con las siguientes características:

1. ESLint
2. Prettier
3. Babel.js
4. Scripts de construcción, desarrollo y producción



Repaso Javascript

Conceptos básicos

- Es case-sensitive
- Las declaraciones acaban en ';' (**ASI**, Automatic Semicolon Insertion)
- Comentarios: `//`, `/**/`
- Hashbang: `#!/usr/bin/env node`

Declaraciones

var: Declara una variable

let: Declara una variable local al ámbito de bloque más cercano

const: Declara una variable constante de solo lectura y ámbito de bloque

- Una variable declarada y no inicializada recibe el valor de **undefined**
- Una variable no declarada arroja el error **ReferenceError**
- Elevación de variables (**hoisting**)



Repaso Javascript

Tipos de datos

Boolean: true / false

null: Valor nulo

undefined: valor no definido

Number: Número entero o en punto flotante

BigInt: Número entero grande

String: Secuencia de caracteres

Symbol: Dato especial con instancias únicas e inmutables

Object: Contenedor de los tipos anteriores

Function: Es un Object que es llamable

Array: Es un Object con claves numéricas y ordenadas



Repaso Javascript

Tipado

- El tipado es dinámico
- Una variable adquiere el tipo del valor que se le asigna
- Una concatenación (+) entre número y cadena se resuelve como cadena
- Conversión a numérico: `parseInt()`, `parseFloat()`, `+'1.1'`

Literales

- Arreglos: `[]`
- numéricos: `5`, `015`, `0xFF`, `0b101`, `3.14`, `-.15`, `.1e-5`
- booleanos: `true`, `false`
- Objetos: `{key: value, ..., keyN: valueN}`
- Expresiones Regulares: `/ab+c/`
- Cadenas: `'abcdef'`, ``abcdef${variable}``, `"abcdef"`



Repaso Javascript

Flujos

Condicionales

- if ... else (false, undefined, null, 0, NaN, “”)
- switch
- throw / try-catch-finally

Bucles

- for
- do...while
- while
- for...in
- for...of

Repaso Javascript

Operadores

Operadores de asignación compuestos

Nombre	Operador abreviado	Significado
<u>Asignación</u>	<code>x = y</code>	<code>x = y</code>
<u>Asignación de adición</u>	<code>x += y</code>	<code>x = x + y</code>
<u>Asignación de resta</u>	<code>x -= y</code>	<code>x = x - y</code>
<u>Asignación de multiplicación</u>	<code>x *= y</code>	<code>x = x * y</code>
<u>Asignación de división</u>	<code>x /= y</code>	<code>x = x / y</code>
<u>Asignación de residuo</u>	<code>x %= y</code>	<code>x = x % y</code>
<u>Asignación de exponenciación</u>	<code>x **= y</code>	<code>x = x ** y</code>

Repaso Javascript

Operadores

Operadores lógicos		
Operador	Uso	Descripción
<u>AND Lógico</u> (&&)	<code>expr1 && expr2</code>	Devuelve <code>expr1</code> si se puede convertir a <code>false</code> ; de lo contrario, devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>&&</code> devuelve <code>true</code> si ambos operandos son <code>true</code> ; de lo contrario, devuelve <code>false</code> .
<u>OR lógico</u> ()	<code>expr1 expr2</code>	Devuelve <code>expr1</code> si se puede convertir a <code>true</code> ; de lo contrario, devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code> </code> devuelve <code>true</code> si alguno de los operandos es <code>true</code> ; si ambos son falsos, devuelve <code>false</code> .
<u>NOT lógico</u> (!)	<code>!expr</code>	Devuelve <code>false</code> si su único operando se puede convertir a <code>true</code> ; de lo contrario, devuelve <code>true</code> .

Repaso Javascript

Operadores

Operadores de comparación

<u>Operador</u>	<u>Descripción</u>	<u>Ejemplos que devuelven true</u>
<u>Igual</u> (==)	Devuelve <u>true</u> si los operandos son iguales.	<u>3 == var1</u> <u>"3" == var1</u> <u>3 == '3'</u>
<u>No es igual</u> (!=)	Devuelve <u>true</u> si los operandos <u>no</u> son iguales.	<u>var1 != 4</u> <u>var2 != "3"</u>
<u>Estrictamente igual</u> (===)	Devuelve <u>true</u> si los operandos son iguales y del mismo tipo. Consulta también <u>Object.is</u> y <u>similitud en JS</u> .	<u>3 === var1</u>
<u>Desigualdad estricta</u> (!==)	Devuelve <u>true</u> si los operandos son del mismo tipo pero no iguales, o son de diferente tipo.	<u>var1 !== "3"</u> <u>3 !== '3'</u>
<u>Mayor que</u> (>)	Devuelve <u>true</u> si el operando izquierdo es mayor que el operando derecho.	<u>var2 > var1</u> <u>"12" > 2</u>
<u>Mayor o igual que</u> (>=)	Devuelve <u>true</u> si el operando izquierdo es mayor o igual que el operando derecho.	<u>var2 >= var1</u> <u>var1 >= 3</u>
<u>Menor que</u> (<)	Devuelve <u>true</u> si el operando izquierdo es menor que el operando derecho.	<u>var1 < var2</u> <u>"2" < 12</u>
<u>Menor o igual</u> (<=)	Devuelve <u>true</u> si el operando izquierdo es menor o igual que el operando derecho.	<u>var1 <= var2</u> <u>var2 <= 5</u>

Repaso Javascript

Operadores

Operadores aritméticos		
Operador	Descripción	Ejemplo
<u>Residuo</u> (%)	Operador binario. Devuelve el resto entero de dividir los dos operandos.	12 % 5 devuelve 2.
<u>Incremento</u> (++)	Operador unario. Agrega uno a su operando. Si se usa como operador prefijo (++x), devuelve el valor de su operando después de agregar uno; si se usa como operador sufijo (x++), devuelve el valor de su operando antes de agregar uno.	Si x es 3, ++x establece x en 4 y devuelve 4, mientras que x++ devuelve 3 y , solo entonces, establece x en 4.
<u>Decremento</u> (--)	Operador unario. Resta uno de su operando. El valor de retorno es análogo al del operador de incremento.	Si x es 3, entonces --x establece x en 2 y devuelve 2, mientras que x-- devuelve 3 y, solo entonces, establece x en 2.
<u>Negación unaria</u> (-)	Operador unario. Devuelve la negación de su operando.	Si x es 3, entonces -x devuelve -3.
<u>Positivo unario</u> (+)	Operador unario. Intenta convertir el operando en un número, si aún no lo es.	+"3" devuelve 3 . +true devuelve 1.
<u>Operador de exponenciación</u> (**)	Calcula la base a la potencia de exponente , es decir, $base^{exponente}$	2 ** 3 returns 8 . 10 ** -1 returns 0.1 .

Repaso Javascript

Operadores

Operadores bit a bit

Operador	Uso	Descripción
<u>AND a nivel de bits</u>	$a \& b$	Devuelve un uno en cada posición del bit para los que los bits correspondientes de ambos operandos son unos.
<u>OR a nivel de bits</u>	$a b$	Devuelve un cero en cada posición de bit para el cual los bits correspondientes de ambos operandos son ceros.
<u>XOR a nivel de bits</u>	$a \wedge b$	Devuelve un cero en cada posición de bit para la que los bits correspondientes son iguales. [Devuelve uno en cada posición de bit para la que los bits correspondientes son diferentes].
<u>NOT a nivel de bits</u>	$\sim a$	Invierte los bits de su operando.
<u>Desplazamiento a la izquierda</u>	$a \ll b$	Desplaza a en representación binaria b bits hacia la izquierda, desplazándose en ceros desde la derecha.
<u>Desplazamiento a la derecha de propagación de signo</u>	$a \gg b$	Desplaza a en representación binaria b bits a la derecha, descartando los bits desplazados.
<u>Desplazamiento a la derecha de relleno cero</u>	$a \ggg b$	Desplaza a en representación binaria b bits hacia la derecha, descartando los bits desplazados y desplazándose en ceros desde la izquierda.

Repaso Javascript

Operadores

- Operador de cadena (+)
- Operador condicional o ternario (exp ? true : false)
- Operador coma (,)
- Operador **delete**
- Operador **typeof**
- Operador **void**
- Operador **in**
- Operador **instanceOf**
- Operador **new**
- Operador **super**

```
typeof myFun;      // devuelve "function"
typeof shape;      // devuelve "string"
typeof size;       // devuelve "number"
typeof foo;        // devuelve "object"
typeof today;      // devuelve "object"
typeof doesntExist; // devuelve "undefined"
```

```
delete object.property;
delete object[propertyKey];
delete objectName[index];
```

```
var theDay = new Date(1995, 12, 17);
if (theDay instanceof Date) {
    // instrucciones a ejecutar
}
```

```
var status = (age >= 18) ? 'adult' : 'minor';
```

```
var mycar = { make: 'Honda', model: 'Accord', year: 1998 };
'make' in mycar; // devuelve true
'model' in mycar; // devuelve true
```




Repaso Javascript

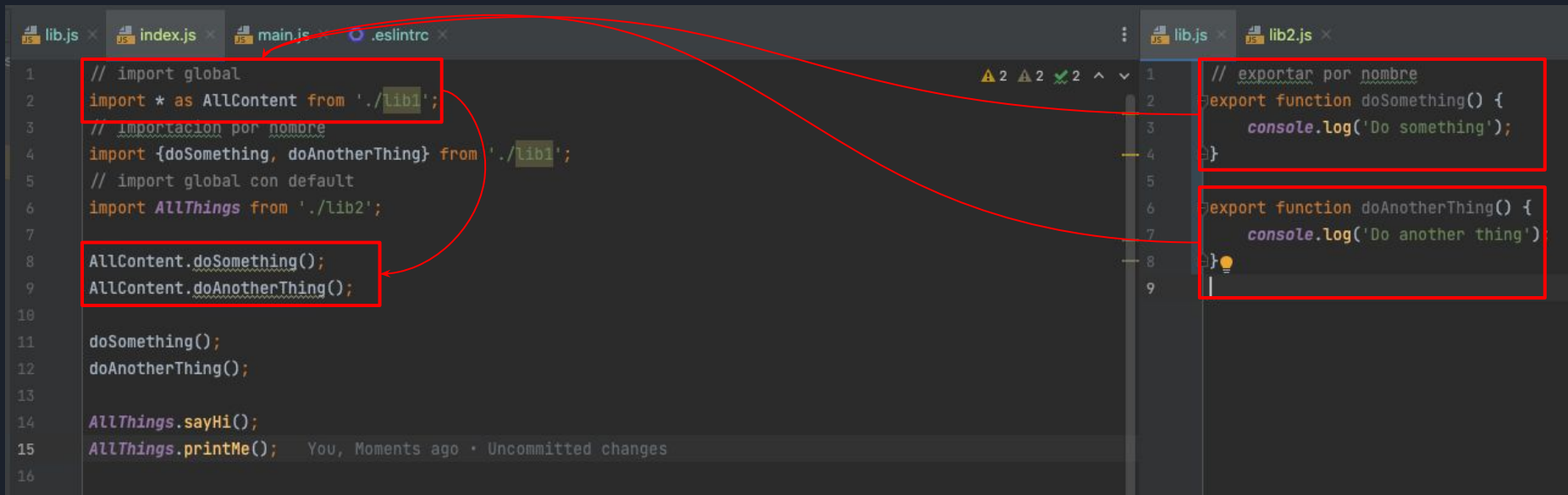
Ejercicios

1. Escribir un algoritmo que muestre los cuadrados de los primeros 10 números
2. Escribir un algoritmo que recorra el rango 1 al 20 y muestre para cada número si es par o impar
3. Escribir un algoritmo que almacene los días de la semana y su traducción al inglés y los muestre por pantalla
4. Escribir un CLI que pida un día de la semana y devuelva su traducción al inglés
5. Escribir un CLI que pida un mes y devuelva el número de días que tiene dicho mes
6. Escribir un CLI que pida un año e indique si es bisiesto
7. Controlar los posibles errores en el CLI, como que no se inserte un número o un día de la semana.

Repaso Javascript

Modulos

Usamos import para importar un módulo y export para exportar.
Podemos exportar todo el contenido del módulo y solo partes de dicho módulo.



The screenshot shows a code editor with two files: `lib.js` and `lib2.js`. Red boxes and arrows highlight the relationship between import and export statements.

```
lib.js
1 // import global
2 import * as AllContent from './lib1';
3 // importacion por nombre
4 import {doSomething, doAnotherThing} from './lib1';
5 // import global con default
6 import AllThings from './lib2';
7
8 AllContent.doSomething();
9 AllContent.doAnotherThing();
10
11 doSomething();
12 doAnotherThing();
13
14 AllThings.sayHi();
15 AllThings.printMe();
```

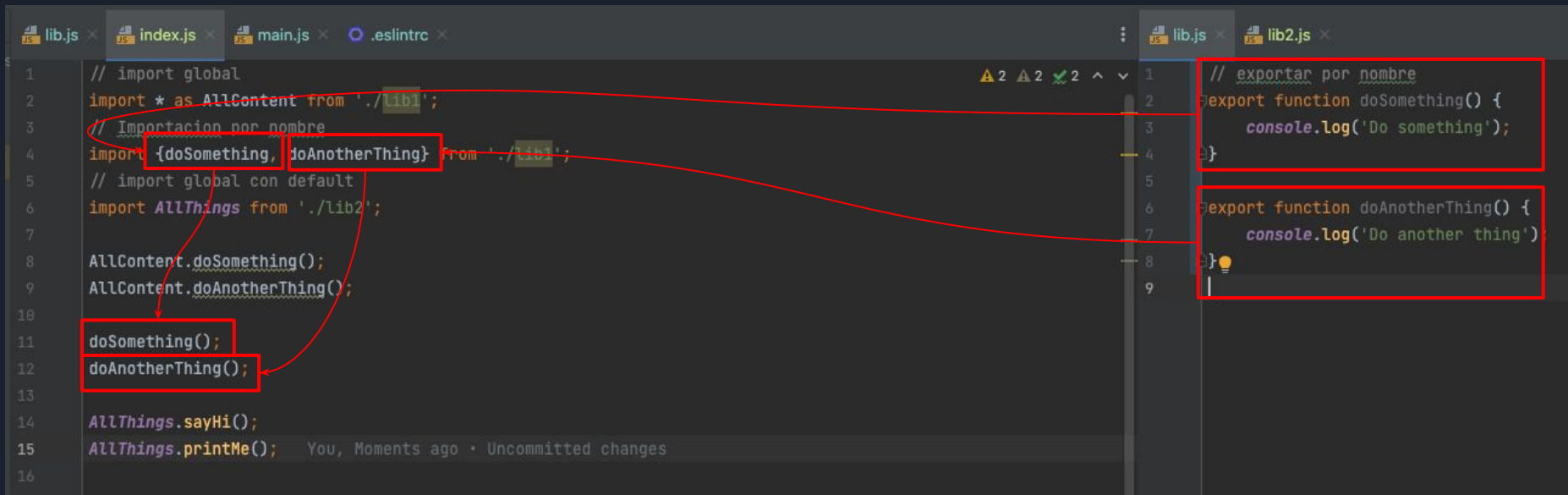
```
lib2.js
1 // exportar por nombre
2 export function doSomething() {
3   console.log('Do something');
4 }
5
6 export function doAnotherThing() {
7   console.log('Do another thing')
8 }
9
```

Annotations in the image:

- A red box around line 2 of `lib.js` (`import * as AllContent from './lib1';`) has an arrow pointing to line 8 of `lib.js` (`AllContent.doSomething();`).
- A red box around line 4 of `lib.js` (`import {doSomething, doAnotherThing} from './lib1';`) has an arrow pointing to line 9 of `lib.js` (`AllContent.doAnotherThing();`).
- A red box around line 2 of `lib2.js` (`export function doSomething() {`) has an arrow pointing to line 8 of `lib.js` (`AllContent.doSomething();`).
- A red box around line 6 of `lib2.js` (`export function doAnotherThing() {`) has an arrow pointing to line 9 of `lib.js` (`AllContent.doAnotherThing();`).

Repaso Javascript

Módulos



The image shows a code editor with two files: `index.js` and `lib2.js`. Red boxes and arrows highlight the relationship between imports and exports.

index.js

```
1 // import global
2 import * as AllContent from './lib1';
3 // Importacion por nombre
4 import {doSomething, doAnotherThing} from './lib2';
5 // import global con default
6 import AllThings from './lib2';
7
8 AllContent.doSomething();
9 AllContent.doAnotherThing();
10
11 doSomething();
12 doAnotherThing();
13
14 AllThings.sayHi();
15 AllThings.printMe();
```

lib2.js

```
1 // exportar por nombre
2 export function doSomething() {
3   console.log('Do something');
4 }
5
6 export function doAnotherThing() {
7   console.log('Do another thing')
8 }
9
```

Annotations:

- Red box around `import {doSomething, doAnotherThing} from './lib2';` in `index.js` (line 4).
- Red box around `doSomething();` in `index.js` (line 11).
- Red box around `doAnotherThing();` in `index.js` (line 12).
- Red box around `export function doSomething() { ... }` in `lib2.js` (lines 2-4).
- Red box around `export function doAnotherThing() { ... }` in `lib2.js` (lines 6-8).
- Red arrows pointing from the import statement in `index.js` to the corresponding function calls in `index.js` and the function definitions in `lib2.js`.

Repaso Javascript

Módulos



```
lib.js x index.js x main.js x .eslintrc x
1 // import global
2 import * as AllContent from './lib1';
3 // Importación por nombre
4 import {doSomething, doAnotherThing} from './lib1';
5 // import global con default
6 import AllThings from './lib2';
7
8 AllContent.doSomething();
9 AllContent.doAnotherThing();
10
11 doSomething();
12 doAnotherThing();
13
14 AllThings.sayHi();
15 AllThings.printMe();
16

lib.js x lib2.js x
1 function printMe() {
2   console.log('Hello World');
3 }
4
5 function sayHi() {
6   console.log('Hi everyone!');
7 }
8
9 // Exportación por defecto
10 export default {
11   printMe,
12   sayHi
13 };
14
15
```

Repaso Javascript

Módulos

```
// import global
import * as AllContent from './lib1';
// Importacion por nombre
import {doSomething, doAnotherThing} from './lib1';

// import global con default y por nombre
import AllThings, {printMe, sayHi} from './lib2';
```

```
AllThings.sayHi();
AllThings.printMe();
```

```
sayHi();
```

```
printMe();
```

```
export function printMe() {
  console.log('Hello World');
}
```

```
export function sayHi() {
  console.log('Hi everyone!');
}
```

```
// Exportacion por defecto
export default {
  printMe,
  sayHi
};
```



Repaso Javascript

Funciones

Son Objetos que pueden ser llamados usando ()

Pueden llamarse a sí mismas. Recursión

Declaración de función

1. Nombre
2. Parámetros ()
3. Cuerpo de la función {}
4. Valor devuelto **return**

Expresión de función

- Son funciones
- No les afecta el hoisting
- Se pueden redefinir
- Pueden ser anónimas, el nombre es opcional
- Se suelen usar como parámetro de otras funciones (Callbacks)



Repaso Javascript

Funciones

Ámbito de una función. (Scope):

- Cada función declara su ámbito
- Una función tiene acceso a su ámbito y al ámbito de todas las funciones que la engloban
- El ámbito global es el que engloba toda la aplicación

Búsqueda de variables:

- Primero busca en el ámbito donde se llama
- Si no la encuentra busca en el ámbito que la engloba
- La búsqueda termina cuando encuentra la variable o alcanza el ámbito global
- Si no la encuentra lanza un ReferenceError. (**strict mode**)
- Un ámbito puede ocultar una variable de un ámbito superior. (**Shadowing**)

Repaso Javascript

Funciones

```
1
2  const globalScope = 'Global Scope';
3
4  export function scopes() {
5
6      ....const mainFunctionScope = 'Main Function';
7      ....const shadowed = 'Shadowed in Main Function';
8
9      ....function innerFunction() {
10
11          ....const innerFunctionScope = 'Inner Function Scope';
12          ....const shadowed = 'Shadowed in Inner Function';
13
14          ....function deepFunction() {
15
16              ....const deepFunctionScope = 'Deep Fuction Scope';
17              ....console.log(
18                  ....deepFunction,
19                  ....innerFunction,
20                  ....mainFunctionScope,
21                  ....globalScope,
22                  ....shadowed
23              );
24          }
25      }
26  }
27
28
```




Repaso Javascript

Funciones

Cierres. Closures. Anidación

- Podemos anidar funciones dentro de otras funciones
- Las funciones internas forman un cierre
- Las funciones internas tienen acceso al ámbito de las funciones que las engloban
- JS recuerda el contenido de un ámbito mientras sea accesible
- Al devolver una función interna esta mantiene el acceso a los ámbitos que la engloban
- Patrón **Module / Revealed Module Pattern**
- Patrón **Facade Pattern**
- Funciones parciales: **Currying**



Repaso Javascript

Funciones

Parámetros

- Parámetros predeterminados
- Propiedad **arguments**
- Parámetros **rest**

Función IIFE

- Immediately Invoked Function Expression

Función constructora

- Es una función normal y corriente
- Devuelve un nuevo objeto en cada invocación
- Podemos referenciar al nuevo objeto dentro de la función usando **this**

Función flecha =>

- Son funciones anónimas
- Ideales para usar en callbacks
- Sin llaves hacen un **return implícito** de la única expresión que pueden contener
- No tienen **this**, **arguments**, **super** ni **new.target**



Repaso Javascript

Ejercicios

Crear un módulo que nos permita gestionar los usuarios de nuestra aplicación mediante una API sencilla. Los usuarios se componen de la propiedad nombre y edad.

1. Con un método para crear un usuario
2. Con un método para listar los usuarios creados
3. Con un método para eliminar un usuario
4. Con un método para buscar un usuario

Añadir al API la función **forEach**, esta función recibirá un callback de usuario, la función iterate recorrerá todos los usuarios almacenados en nuestro módulo y llamará a la función callback tantas veces como usuarios haya almacenados usando cada vez a un usuario como argumento.

Añadir al módulo la función **toString**, esta función usará iterate para mostrar un listado bien formateado de los usuarios almacenados

Crear un CLI que nos permita hacer uso del módulo de usuario.



Repaso Javascript

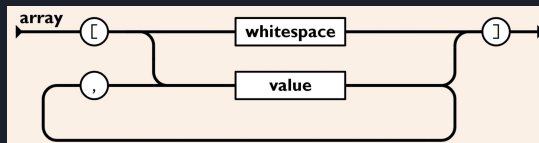
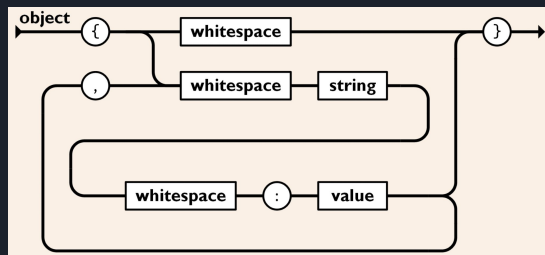
Objetos

- Son un contenedor de propiedades referenciadas mediante los pares clave / valor
- Podemos asignar propiedades en la declaración
- Podemos crear un objeto usando **new Object()**
- Podemos crear un objeto usando una función constructor **'new'**
- Podemos declarar getters y setters
- Podemos asignar/acceder propiedades mediante el operador **'.'**
- Podemos asignar/acceder propiedades mediante el operador **'[]'**
- Podemos asignar propiedades mediante **Object.defineProperty()**
- Usamos **this** para referenciar las propiedades del objeto
- Podemos eliminar propiedades con **delete**
- Podemos usar el iterador **for...in** para iterar sobre las claves del objeto

Repaso Javascript

Objetos

- JavaScript Object Notation. JSON



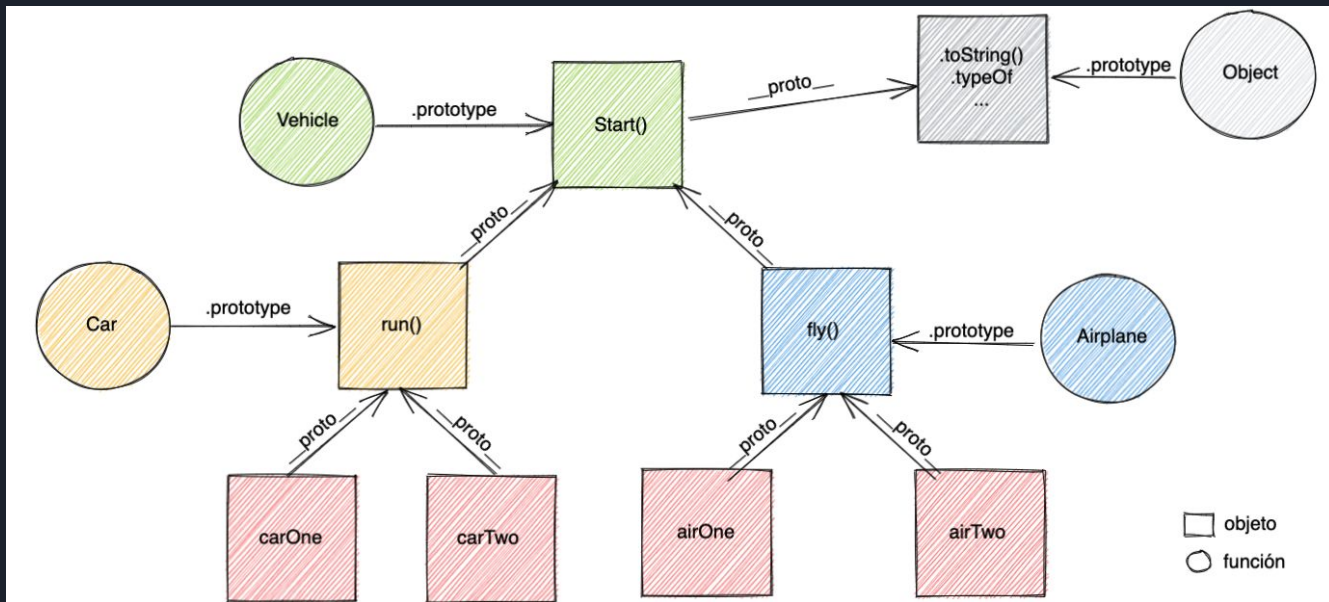
- Serializar objetos con **JSON.stringify()**
- Deserializar objetos con **JSON.parse()**
- import MyObject from './file.json'

```
[
  {
    "peugeot": {
      "model": "5008",
      "kms": 101500,
      "date": "2010-05-19T10:00:00.000Z"
    },
    {
      "chevrolet": {
        "model": "Aveo",
        "kms": 40100,
        "date": "2014-11-19T11:00:00.000Z"
      },
      {
        "renault": {
          "model": "Megane",
          "kms": 75400,
          "date": "2006-10-19T12:00:00.000Z"
        },
        {
          "citroen": {
            "model": "Xsara",
            "kms": 234990,
            "date": "2004-03-19T13:00:00.000Z"
          }
        }
      }
    }
  ]
```

Repaso Javascript

Clases: Objetos + Funciones

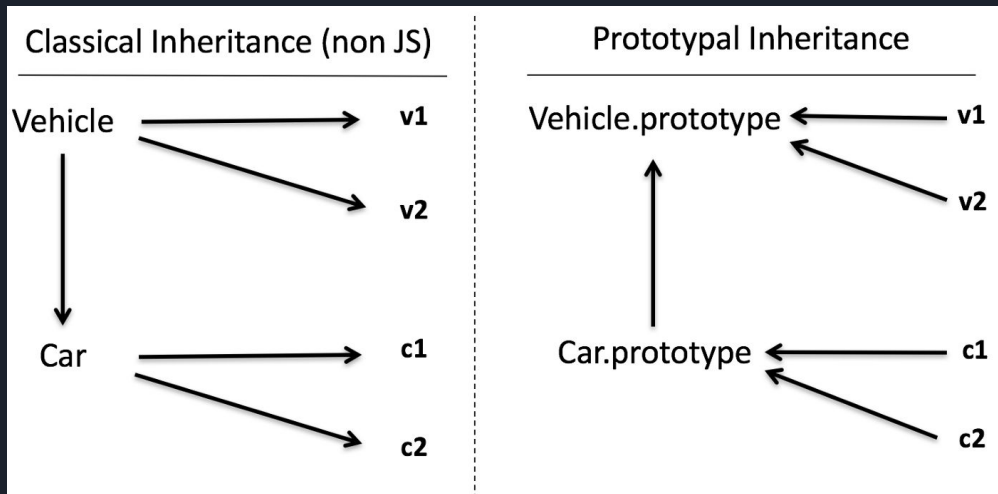
- JavaScript usa herencia de prototipo o por delegación
- Propiedad `__proto__` en los objetos
- Propiedad `prototype` en las funciones



Repaso Javascript

Clases: Objetos + Funciones

- Es una ayuda sintáctica para modelar la herencia de prototipo
- No añade un nuevo sistema de herencia
- No hay fase de copiado
- No se crean dos entidades separadas





Repaso Javascript

Clases: ES6

- Son funciones constructoras
- Se declaran con la palabra reservada class
- No se les aplica hoisting
- Podemos declarar clases anónimas (Expresiones de clase)
- Pueden tener el método “constructor” para inicializar el objeto creado
- Pueden contener métodos de instancia
- Pueden contener métodos estáticos
- Podemos heredar de otra clase con “extends”
- Podemos llamar a métodos de la clase padre usando super
- No se permite la herencia múltiple



Repaso Javascript

Ejercicios

Transformar el ejercicio anterior para usar clases en lugar de el patrón módulo

- Crear la clase Users
- Crear la clase UserDB

Ampliar la funcionalidad de UserDB permitiendo persistir la información en un fichero JSON



Repaso Javascript

Colecciones

- Arrays: Lista ordenada de valores indexados
- Inicializar arreglos
- Tamaño de un array: length
- Acceder a elementos por su posición
- Modificar valor de una posición
- Recorrer arreglos.
- Concatenar arreglos.
- Concatenar elementos de un arreglo
- Añadir un elemento al final del arreglo
- Eliminar el último elemento del arreglo
- Añadir elementos al principio del arreglo
- Eliminar el primer elemento de un arreglo



Repaso Javascript

Colecciones

- Seccionar un arreglo
- Girar un arreglo
- Ordenar un arreglo
- Buscar en el arreglo
- Filtrar el arreglo
- Comprobar valores del arreglo
- Transformar el arreglo



Repaso Javascript

Colecciones

Mapas: Asociación de pares clave valor

WeakMap: Usa objetos para la clave, para que sea susceptible de ser liberada por el GC

Set: Colecciones de valores únicos

WeakSet: Usa objetos para los valores, para que sean susceptibles de ser liberados por el GC



Repaso Javascript

Ejercicios

- 1 Iniciar un Array con los factoriales del 1 al 10
- 2 Dado el arreglo mix, extraer un array que sólo tenga los objetos del tipo conductor
- 3 Crear un módulo que nos calcule la constante de Conway, también conocido como Look-and-say sequence Para ello vamos a implementar un módulo que reciba un número y lo descomponga en las repeticiones de cada dígito seguido del dígito.
Por ejemplo, 1221 se representará como 112211, 211 como 1221 y 111 como 31
- 4 Crear un CLI que pida una cadena de números y nos devuelva su secuencia Look-and-say

Repaso Javascript

Promesas

- Representa la terminación de una operación asíncrona
- Permite una mejor gestión del código asíncrono que las callbacks
- Añadimos el código asíncrono en el método **then**
- El método **then** tiene dos callbacks, la primera representa que la operación se realizó correctamente y la segunda que se produjo algún error en dicho proceso

```
....// Usando callbacks
....fs.readFile('./package.json', (error, data) => {
....|....if (error) {
....|....|....console.log('Error in file', error);
....|....|....return;
....|....}
....
....|....const dataString = data.toString();
....|....const object = JSON.parse(dataString);
....|....console.log('File', object.name);
....|....});
```

```
....// Usando Promesas
....readFile('./package.json').then(
....|....data => {
....|....|....const dataString = data.toString();
....|....|....const object = JSON.parse(dataString);
....|....|....console.log('File', object.name);
....|....},
....|....error => {
....|....|....console.log('Error in file', error);
....|....}
....);
```



Repaso Javascript

Promesas

- **Garantizan** que sus callbacks nunca serán llamadas antes de que acabe el proceso
- **Garantizan** que sus callbacks serán llamadas incluso aunque se registren después de la terminación del proceso
- **Garantizan** que sus callbacks se ejecutarán una sola vez y solo una de ellas
- Podemos encadenar tantos **then** como necesitemos y se nos **garantiza** que todos serán llamados y en el mismo orden en el que se encadenaron
- Si el proceso de una promesa lanza una excepción, esta recorrerá la cadena de manejadores buscando un manejador de error “**catch**”
- Podemos crear una promesa que envuelva un API basada en callbacks
- Podemos componer flujos de promesas encadenando promesas
- **Async / Await** nos permite usar código asíncrono al estilo secuencial



Repaso Javascript

Ejercicios

1. Crear un CLI que acepte una ruta a un fichero como parámetro y devuelva el número de caracteres del fichero
2. Modificar el ejemplo anterior para aceptar el parámetro -l que devuelva el número de líneas del fichero
3. Crear un CLI que le pasemos unas coordenadas GPS y nos devuelva la información política de dichas coordenadas
4. Crear un CLI que reciba una palabra en inglés y nos devuelva su traducción
5. Crear un CLI que reciba un nombre de usuario de github y muestre la información del usuario
6. Modificar el CLI para que le indiquemos un usuario y el nombre de uno de sus repositorios y nos muestre la información del repositorio



Repaso Javascript

Ejercicios

Nos han pedido construir una juguete para jóvenes programadores, que les permita familiarizarse con las puertas lógicas AND, OR, NOT, LSHIFT y RSHIFT.

El juego consiste en un tablero perforado en el que podemos encajar tres tipos de piezas:

- Puertas lógicas: AND, OR, NOT, LSHIFT y RSHIFT.
- Acumuladores: Son elementos a los que podemos asignar valores enteros de 16 bits.
- Señales: Son elementos que generan valores enteros de 16 bits.

El juego viene provisto de cables que nos permiten unir las puertas lógicas y los acumuladores para crear la lógica que que queramos.

Ejemplo: Señal(24) -> Acumulador(X) Almacena la señal 24 en el acumulador X

Nuestra misión es escribir el software que calculará las operaciones de AND, OR, NOT, LSHIFT y RSHIFT que el jugador vaya construyendo mediante el tablero de juego para mostrarlas en los acumuladores.

Ejemplo: 123 -> x: La señal 123 es almacenada en el acumulador x

Ejemplo: x AND y -> z: El acumulador x se combina con el y mediante la puerta lógica AND y su resultado se almacena en el acumulador z



Repaso Javascript

Ejercicios

Ejemplo de un programa escrito por un joven programador:

```
6 -> a
5 -> b
a OR b -> x
a AND b -> y
x RSHIFT 2 -> w
y LSHIFT 3 -> z
```

La salida del módulo será el estado de cada acumulador usado:

```
{ a: 6, b: 5, w: 1, x: 7, y: 4, z: 32 }
```

Crear un CLI que nos pida instrucciones y calcule el nuevo estado de los acumuladores tras cada instrucción.

Añadir una opción al CLI que nos permita ver el estado de cada acumulador

Añadir una opción al CLI que permita guardar o recuperar el estado actual de los acumuladores