Curso Angular

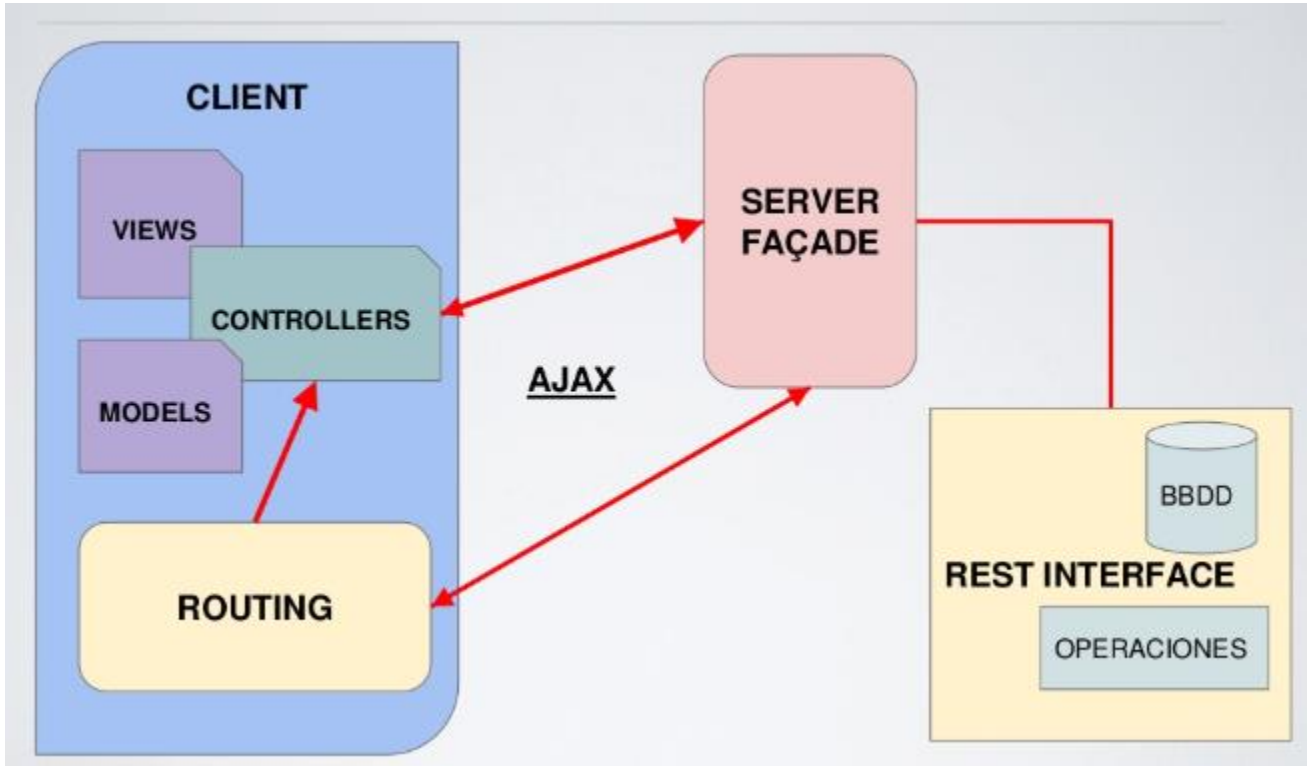
Tema 1. Introducción

Índice

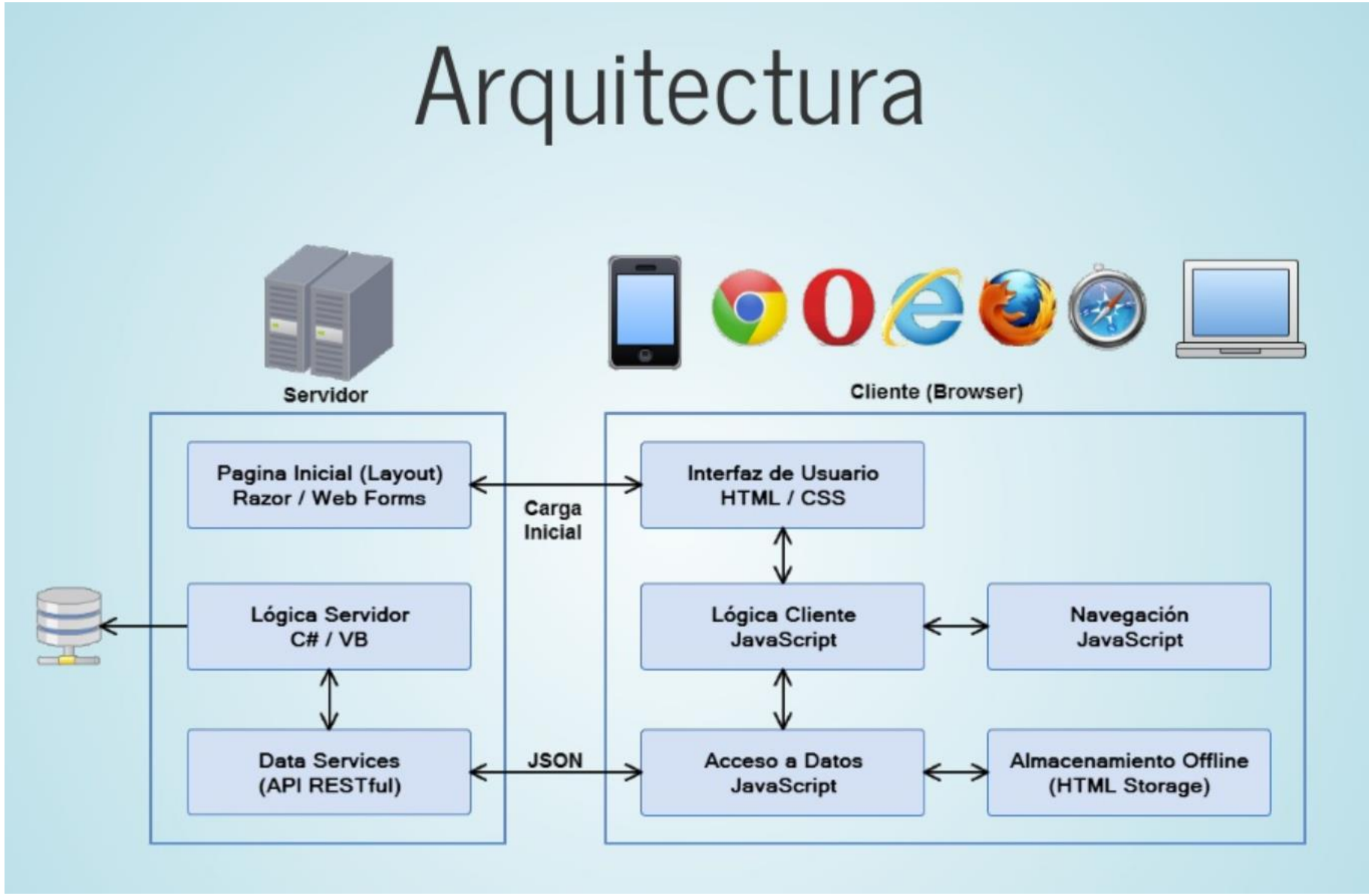
SPA: Single Page Application.

- Es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador.
- Técnicamente: aplicación con un único punto de entrada.
- Solo existe un único punto de entrada, y a través de él se pueden acceder a diferentes vistas.
  - En el mismo punto de entrada, se irán intercambiando distintas vistas, produciendo el efecto de que tienes varias páginas, cuando realmente todo es la misma, intercambiando vistas.
- Ejemplo: GMAIL

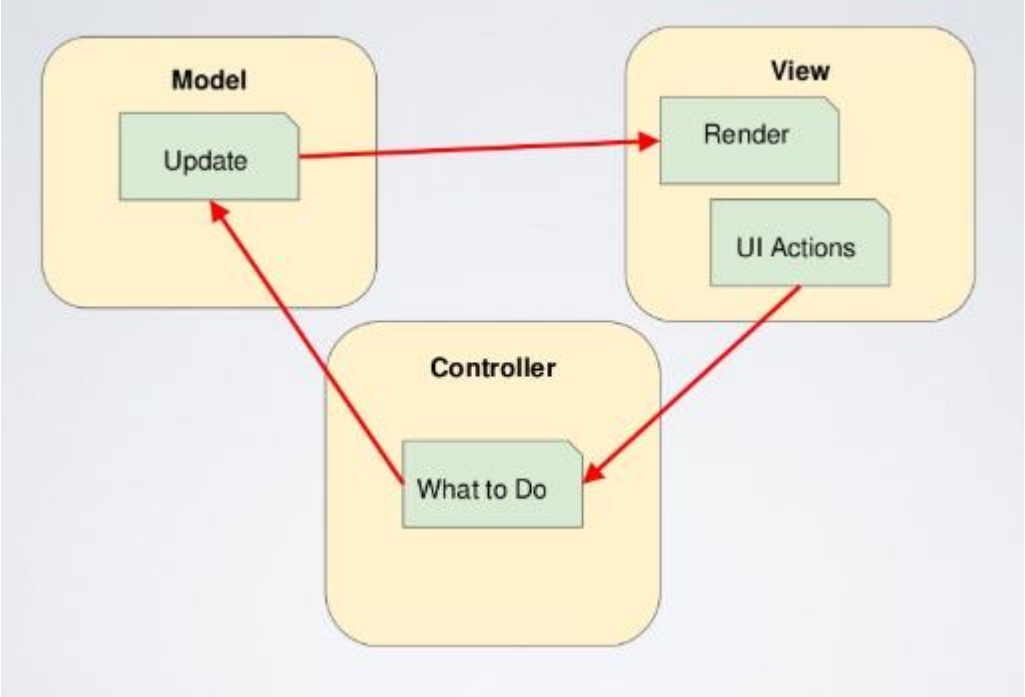
Bloques SPA


CLIENT
VIEWS
CONTROLLERS
MODELS
ROUTING
AJAX
SERVER
FAÇADE
BBDD
REST INTERFACE
OPERACIONES

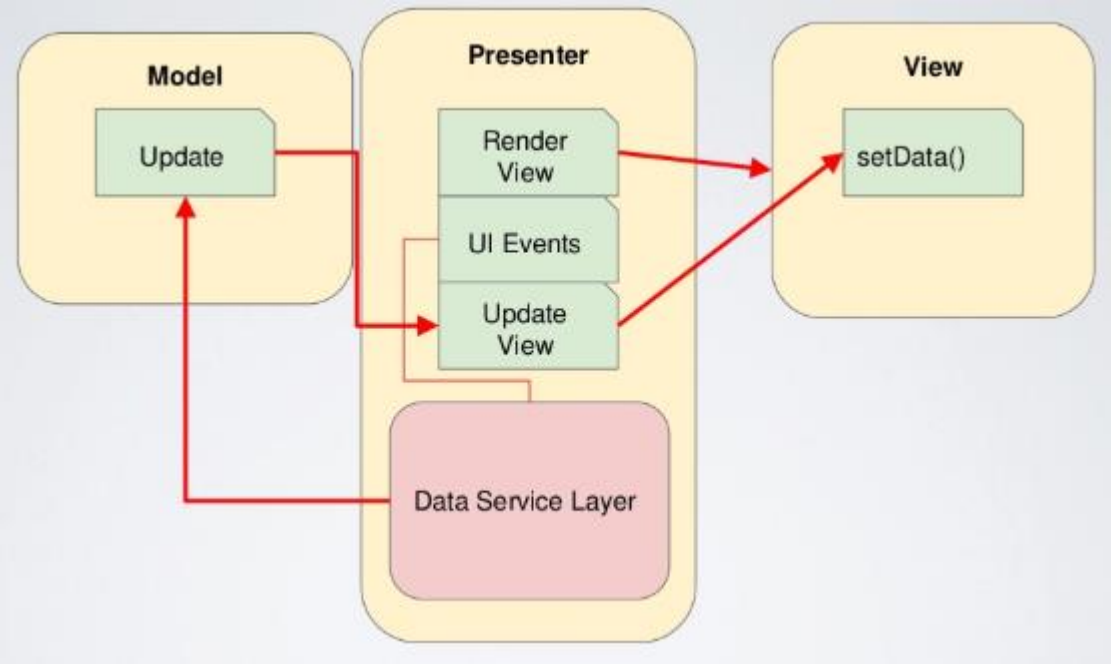
Arquitectura



Modelo MVC



Modelo: Model-View-Presenter



SPA

Para aclarar posibles malentendidos:

SPA no implica que no cambie la dirección de la barra de direcciones, es decir, la URL a la que accedes con el navegador.

De hecho, es normal que al interaccionar con una SPA la URL que se muestra en la barra de direcciones del navegador vaya cambiando

La clave es que, aunque cambie esta URL, la página no se recarga.

El hecho de cambiar esa URL es algo importante, ya que el propio navegador mantiene un historial de pantallas entre las que el usuario se podría mover, pulsando el botón de "atrás" en el navegador o "adelante".

Con todo ello, se consigue que el usuario pueda usar el historial como lo hace en una página normal, siendo que en una SPA la navegación entre la secuencia de páginas del historial se realiza realmente entre vistas de la aplicación.

Problemas con SPA

- Alta complejidad técnica
- Sistema de ***routing*** adaptado a las necesidades de la aplicación
- Uso de ***analytics*** más complejo por la navegación sintética
- Los diseñadores no están preparados técnicamente y deben trabajar sobre maquetas que después son integradas por los programadores
- No amigable para robots y spiders de los buscadores
- El navegador debe tener activado JS
- Falta de sincronismo con el servidor en navegación tradicional

Lenguajes y tecnologías para producir una SPA

Una SPA se implementa utilizando Javascript.

No existe ningún otro tipo de lenguaje en el que puedas realizar una SPA, ya que básicamente es una aplicación web que se ejecuta del lado del cliente.

Nota: Obviamente, también se realiza usando HTML + CSS, para la presentación, ya que son los lenguajes que entiende el navegador.

Existen diversas librerías y frameworks que facilitan el desarrollo de una SPA:

- AngularJS
- Angular 2
- React
- Polymer
- EmberJS

URL: Universal Resource Locator (Localizador Uniforme de Recursos)

- Describen la localización de un recurso (documento o una página)
- Anatomía de una URL

Los fragmentos se utilizan en las URLs para saltar a una sección específica dentro de una página.

ES6 JavaScript & TypeScript

Transpilación

- El navegador no interpreta TypeScript.
- Convertir TypeScript a JavaScript
- Actualmente, los navegadores soportan JavaScript ES5
- Transpilación: Proceso de convertir TypeSCript a Javascript
- Uso de un compilador: tsc
  - npm install -g typescript
- Verificar la versión: tsc -v
- Para crear un fichero de configuración: tsc --init

TypeScript

- Lenguaje de programación ***open source creado por Microsoft***
- Primera versión liberada en 2012
- TypeScript es un superconjunto de ES6

ES6

- ES6 es un acuerdo de como debería funcionar JavaScript en el futuro
- Cada una de las empresas que implementan navegadores deciden como implementar las características respetando el acuerdo
- A día de hoy Chrome es el navegador que da el mejor soporte a ES6
- En ocacsiones las características ES6 se activan cuando se utiliza se indica el modo “use sctrict”.
- Recomendación: agregar “use strict” a los ficheros
- Para compilar con las características: tsc -t ES6 -w nombreFichero.ts

Visita este enlace para ver las características ES6 y el soporte: http://kangax.github.io/compat-table/es6/

LET

Un punto confuso para los desarrolladores procedentes de diferentes lenguajes de programación es el comportamiento de las variables teniendo en cuenta el ámbito.

- El ámbito (Scope) se refiere al ciclo de vida de una variable: Cuando y hasta cuando es visible una variable.
- En java y en C++ existe el concepto de “ámbito de bloque”
  - Un bloque es código que está envuelto entre llaves: { }

```
{
        // Este es un bloque
}
// Esto ya no forma parte del bloque
```

En estos lenguajes si se declara una variable dentro del bloque, dicha variable solo es visible dentro del bloque

- Pero en ES5 sólo existen 2 ámbitos: “ámbito global” y “ámbito de función”

```
{
        var a = "block";
 }
console.log(a);
```

IIFE (Immediately Invoked Function Expression)

```
function hello() {
  var a = "function";
  for (var i=0; i<5; i++) {
        (function() {
        var a = "block";
  })();
  }
  console.log(a);
}
hello();
```

```
function hello() {
  var a = "function";
  for (var i=0; i<5; i++) {
        function something() {
        var a = "block";
  };
  }
  console.log(a);
}
hello();
```

¿Cual es la salida obtenida al ejecutar el código?

"function"

Dado que las funciones tienen su propio ámbito, utilizando IIFE se tiene el mismo efecto a tener ámbito a nivel de bloque.

La variable a dentro del IIFE no es visible fuera del IIFE.

Let

IIFE funciona, pero es una forma bastante larga de resolver este problema.

Con ES6 ahora tenemos una nueva palabra clave ***let***, la usamos en lugar de la palabra clave ***var*** y sirve para crear una variable ***con nivel de bloque***

```
function hello() {
  var a = "function";
  for (var i = 0; i < 5; i++) {
     let a = "block";
  }
  console.log(a);
}
hello();
```

La variable a declarada dentro del bucle sólo existe dentro del cuerpo de dicho bucle

Uso de let en bucles for

```
var funcs = [];

for (var i = 0; i < 5; i += 1) {
  var y = i;
  funcs.push(function () {
  console.log(y);
  })
}

funcs.forEach(function (func) {
  func()
});
```

Adivina: ¿Cuál es la salida de este fragmento de código?

a)
0
1
2
3
4
5

b)
5
5
5
5
5
5

```
var funcs = [];

for (var i = 0; i < 5; i += 1) {
  let y = i;
  funcs.push(function () {
  console.log(y);
  })
}

funcs.forEach(function (func) {
  func()
});
```

Bucle for

```
var funcs = [];

for (let i = 0; i < 5; i += 1) {
  funcs.push(function () {
  console.log(i);
  })
}

funcs.forEach(function (func) {
  func()
});
```

A pesar de que let i = 0 está estrictamente declarado fuera del bloque for {}

Cualquier variable declarada en la ***expresión de bucle*** con let tiene alcance de nivel de bloque en el bucle!

Resumen

La palabra reservada let es muy importante en ES6

No es un reemplazo de var.

var puede seguir siendo utilizada en ES6, con la misma semántica de ES5

Recomendación: utilizar let

Const

- Nueva palabra clave
- Declara una variable como “constante”
- const se utiliza para declarar variables, pero a diferencia de var y let, la variable debe ser inmediatamente declarada
- Tanto let como const crean variables con ámbito a nivel de bloque. Sólo existen dentro del bloque en el que han sido declaradas.

```
function func() {
  if (true) {
    const tmp = 123;
  }
  console.log(tmp); // Error
}
func();
```

Template Strings

Multi-Lines Strings: Con ES5 y ES6 se pueden declarar cadenas de caracteres colocándolos entre ‘ o entre

let single = "hello world";

Si se necesita que la cadena se extienda a varias líneas:

En ES5:

```
let single = 'hello\n'
+   'world\n'
+   'my\n'
+   'name\n'
+   'is\n'
+   'asim\n';
console.log(multi);
```

En Es6:

```
let multi = `
hello
world
my
name
is
asim`;
console.log(multi);
```

Substitución de variables

En ES6:

```
let name = "asim";

let multi = `
hello
world
my
name
is
${name}
`;
console.log(multi);
```

Funciones flecha

Javascript tiene funciones de primera clase:

Las funciones JavaScript pueden ser pasadas al igual que cualquier otro valor

Podemos pasar una función como argumento a la función setTimeout:

```
setTimeout( function() {
  console.log("setTimeout called!");
}, 1000);
```

La función que hemos pasado como argumento, se denomina función anónima.
ES6 introduce una sintaxis ligeramente diferente para definir funciones anónimas: sintaxis flecha

```
setTimeout(  () => {
  console.log("setTimeout called!")
}, 1000);
```

Si la función sólo contiene una expresión podemos omitir las llaves

```
setTimeout( () => console.log("setTimeout called!") , 1000);
```

Argumrntos en funciones flecha

```
let add = function(a,b) {
  return a + b;
};
```

```
let add = (a,b) => a + b;
```

Destructuring

La desestructuración es una forma de extraer valores en variables a partir de datos almacenados en objetos y vectores

```
const obj = {first: 'Pedro', last: 'Sánchez', age: 45 };
```

Si queremos extraer la primera y la última propiedad en variables locales:

ES5:

```
const f = obj.first;
const l = obj.last;
console.log(f); // Pedro
console.log(l); // Sánchez
```

ES6:

```
const { first: f,  last: l } = obj;
console.log(f); // Pedro
console.log(l); // Sánchez
```

Atajo en ES6

```
// {prop} es equivalente a {prop: prop}
const {first, last} = obj;
console.log(first); // Pedro
console.log(last); // Sánchez
```

{first: f, last: l} describe un patrón. Un conjunto de reglas de cómo queremos desestructurar un objeto

```
const {first: f} = obj;
```
extrae la primer propiedad y la almacena en una constante llamada t

Desestructuración de un vector

```
const arr = ['a', 'b'];
const [x, y] = arr;
console.log(x); // a
console.log(y); // b
```

Iteración

Iteración sobre vectores

```
let array = [1,2,3];
for (let i = 0; i < array.length; i++) {
  console.log(array[i]);
}
```

```
let array = [1,2,3];
array.forEach(function (value) {
  console.log(value);
});
```

1. No se puede romper el bucle
2. No se puede regresar utilizando una sentencia return

For In & For Out

Diseñado para iterar sobre las propiedades de los objetos:

```
var obj = {a:1,b:2};
for (let prop in obj) {
  console.log(prop);
}
// a
// b
```

```
let array = [10,20,30];
for (let index in array) {
  console.log(index);
});
// 0
// 1
// 2
```

```
let array = [10,20,30];
for (let index in array) {
  console.log(typeof(index));
};
// string
// string
// string
```

```
let array = [10,20,30];
for (var value of array) {
  console.log(value);
}
// 10
// 20
// 30
```

Resumen

The for–in loop is for looping over object properties

The for–of loop is for looping over the values in an array

for–of is not just for arrays.

It also works on most array-like objects including the new Set and Map types which we will cover in the next lecture.

Map & Set

Uso de Object como relación clave-valor

```
let obj = {key: "value", a: 1}
console.log(obj.a); // 1
 console.log(obj['key']); // "value"
```

```
let base = {a:1,b:2};
let obj = Object.create(base);
obj[c] = 3;
for (prop in obj) console.log(prop)
// a
// b
// c
```

```
let base = {a:1,b:2};
let obj = Object.create(base);
obj[c] = 3;
for (prop in obj) {
  if (obj.hasOwnProperty(prop)) {
  console.log(prop)
  }
}
// c
```

Map

Nueva estructura de datos introducida en ES6

```
let map = new Map();
map.set("A",1);
map.set("B",2);
map.set("C",3);
```

```
let map = new Map()
  .set("A",1)
  .set("B",2)
  .set("C",3);
```

```
let map = new Map([
  [ "A", 1 ],
  [ "B", 2 ],
  [ "C", 3 ]
]);
```

```
map.get("A");
// 1
```

```
map.has("A");
// true
```

```
map.delete("A");
// true
```

```
map.size
// 2
```

```
map.clear()
map.size
// 0
```

```
for (let key of map.keys()) {
  console.log(key);
}
```

```
for (let value of map.values()) {
  console.log(value);
}
// 1:
// 2
// 3
```

```
for (let entry of map.entries()) {
  console.log(entry[0], entry[1]);
}
// "APPLE" 1
// "ORANGE" 2
// "MANGO" 3
```

Utilizando desestructuración podemos accede a las claves y a los valores de forma directa:

```
for (let [key, value] of map.entries()) {
  console.log(key, value);
}
// "APPLE" 1
// "ORANGE" 2
// "MANGO" 3
```

```
for (let [key, value] of map) {
  console.log(key, value);
}
// "APPLE" 1
// "ORANGE" 2
// "MANGO" 3
```

Set

Equivalente a Map, en este caso únicamente se almacena un valor

```
let set = new Set();
set.add('APPLE');
set.add('ORANGE');
set.add('MANGO');
```

```
let set = new Set()
  .add('APPLE')
  .add('ORANGE')
  .add('MANGO');
```

```
let set = new Set(['APPLE', 'ORANGE', 'MANGO']);
```

```
set.has('APPLE')
// true
```

```
set.delete('APPLE')
```

```
set.size
// 2
```

```
set.clear();
set.size
// 0
```

```
let set = new Set();
set.add('Moo');
set.size
// 1
set.add('Moo');
set.size
// 1
```

```
let set = new Set(['APPLE', 'ORANGE', 'MANGO']);
for (let entry of set) {
  console.log(entry);
}
// APPLE
// ORANGE
// MANGO
```