Tema 2.2
Funciones

- Cómo crear funciones
- Funciones anónimas
- Funciones autoejecutables
- Funciones clausura
- Callbacks
- HOF
- Función Arrow

Funciones

Las **funciones** son uno de los tipos de datos más importantes, ya que estamos continuamente utilizándolas a lo largo de nuestro código.
En Javascript, las funciones pueden ser tipos de datos como un número o una cadena de texto .

```
typeof function () {}; // 'function‘
```

Formas de crear funciones:

Ejemplo	Descripción
`function nombre(p1, p2...) { }`	Mediante **declaración** (la más usada por principiantes)
`let nombre = function(p1, p2...) { }`	Mediante **expresión** (la más habitual en programadores con experiencia)
`new Function(p1, p2..., code);`	Mediante un constructor de **objeto** (no recomendada)

Cómo crear funciones

Funciones por declaración

La forma más popular, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación.
Esta forma permite declarar una función que existirá a lo largo de todo el código:

```
function saludar() {
        return "Hola";
}
saludar(); // 'Hola'
typeof saludar; // 'function‘
```

Podríamos ejecutar la función saludar() incluso antes de haberla creado y funcionaría correctamente, ya que Javascript primero busca las declaraciones de funciones y luego procesa el resto del código.

Cómo crear funciones

Funciones por expresión

Consiste en «guardar funciones» dentro de variables, para posteriormente «ejecutar dichas variables».

```
const saludo = function saludar() {
        return "Hola";
        };
saludo(); // 'Hola'
```

Así, creamos una función **en el interior de una variable**, lo que nos permitirá posteriormente ejecutar la variable (*como si fuera una función, que de hecho lo es*).

El nombre de la función (*saludar*) pasa a ser inútil. Si ejecutamos saludar() nos dirá que no existe y ejecutamos saludo() funciona correctamente. Ahora el nombre de la variable pasa a ser el «nombre de la función», mientras que el anterior nombre de la función desaparece y se omite, creando un concepto llamado **funciones anónimas** (*o funciones lambda*).

Diferencia entre funciones por declaración y funciones por expresión: éstas sólo están disponibles a partir de la inicialización de la variable. Si «ejecutamos la variable» antes de declararla, nos dará un error.

Cómo crear funciones

Funciones como objetos

Se pueden declarar funciones como si fueran **objetos**.
Sin embargo, esto no se suele utilizar en el mundo real, ya que es incómodo, poco práctico y muy verboso:

```
const saludar = new Function("return 'Hola';");
saludar(); // 'Hola‘
```

Sólo es interesante saberlo para darse cuenta que en Javascript todo pueden ser objetos

Funciones anónimas

Las **funciones anónimas** (*o funciones lambda*) son un tipo de funciones que se declaran sin definir un nombre de función, alojándolas en el interior de una variable y haciendo referencia a dicha variable cada vez que queramos utilizarla.

```
// Función anónima "saludo"
const saludo = function () {
            return "Hola";
            };
saludo(); // 'Hola'
saludo; // ƒ () { return 'Hola'; }
```

Observa que, tras definir la función, hace dos acciones:

- Ejecutamos la variable saludo, que como contiene una función, se ejecuta la función
- Mostramos el valor de la variable saludo (*no indicamos paréntesis, no ejecutamos*)

Funciones Arrow

Las **Arrow functions**, funciones flecha o «fat arrow» son una forma corta y compacta de escribir las funciones tradicionales de Javascript. Elimina la palabra function y añadir => antes de abrir las llaves:

```
const func = function () {
        return "Función tradicional.";
};

const func = () => {
        return "Función flecha.";
};
```

Además:

1. Si el cuerpo de la función sólo tiene una línea, podemos omitir las llaves (*{}*).
2. En ese caso, se hace un return implícito, por lo que podemos omitir también el return.
3. Si la función no tiene parámetros, se indica como en el ejemplo anterior: () =>.
4. Si la función tiene un solo parámetro, opcionalmente te puedes ahorrar los paréntesis-
5. Si la función tiene 2 ó más parámetros, se indican entre paréntesis: (a, b) =>.
6. Si queremos devolver un objeto, que coincide con la sintaxis de las llaves, se puede englobar con paréntesis: ({name: 'Mario}).

Funciones Arrow

Por lo tanto, el ejemplo anterior se puede simplificar aún más:

```
// 0 parámetros: Devuelve "Función flecha"
const func = () => "Función flecha.";
// 1 parámetro: Devuelve el valor de e + 1
const func = (e) => e + 1;
// 2 parámetros: Devuelve el valor de a + b
const func = (a, b) => a + b;
```

Las **funciones flecha** hacen que el código sea mucho más legible y claro de escribir, mejorando la productividad a la hora de escribir nuestro código.