

### Funciones autoejecutables

A veces, se necesita crear una función y ejecutarla inmediatamente, sabiendo que no la vas a volver a necesitar. También se le llama **IIFE** (*Función por expresión invocada inmediatamente*).

Esto se conoce como **función autoejecutable**.

Sólo tenemos que envolver entre paréntesis la función anónima en cuestión (*no necesitamos que tenga nombre, puesto que no la vamos a guardar*) y luego, ejecutarla:

```
(function () {  
    console.log("Hola!!");  
})();
```

Si queremos pasar parámetros, éstos se ponen al final de la función autoejecutable:

```
(function (name) {  
    console.log(`¡Hola, ${name}!`);  
})("Mario");
```

### Funciones autoejecutables

Si la función autoejecutable devuelve algún valor con return, a diferencia de las **funciones por expresión**, en este caso lo que se almacena en la variable es el valor que devuelve la función autoejecutada:

```
const value = (function (name) {  
    return `¡Hola, ${name}!`;  
})("Mario");  
value; // '¡Hola, Mario!'  
typeof value; // 'string'
```

Lo analizamos paso a paso:

- Se ejecuta la función autoejecutable contenida en paréntesis.
- La función toma el valor Mario como parámetro name.
- La función devuelve el valor ¡Hola, Mario! como resultado.
- La variable value almacena el valor devuelto por la función autoejecutada.

### Funciones Clausuras

La **clausura** (o *cierre*) es una función que «encierra» variables en su propio ámbito (y *que continúan existiendo aún habiendo terminado de ejecutar la función*).

Por ejemplo:

```
const incr = (function () {  
    let num = 0;  
    return function () {  
        num++;  
        return num;  
    };  
})();
```

Analizamos paso a paso:

- Tenemos una **función anónima** que es también una función autoejecutable.
- La función autoejecutable devuelve otra función diferente.
- Al ejecutar la función la primera vez (*autoejecutable*) devolvemos esa otra función que es la que se guarda en `incr` para futuras ejecuciones.

### Funciones Clausuras

La «magia» de las clausuras es que en el interior de la función autoejecutable estamos creando una variable num que se guardará en el ámbito de dicha función autoejecutable, por lo tanto seguirá existiendo, inicialmente con el valor declarado: 0.

En la variable incr guardamos una función que además conoce el valor de una variable num, que sólo existe dentro de incr.

```
typeof incr; // 'function'  
incr(); // 1  
incr(); // 2  
incr(); // 3
```

La función que devolvemos, lo que hace es incrementar el valor de num y devolverlo. Como la variable incr es una clausura y mantiene la variable en su propio ámbito, veremos que a medida que ejecutamos incr(), los valores de num (*que estamos devolviendo*) conservan su valor y se van incrementando.

## Callbacks

Un **callback** (*llamada hacia atrás*) es pasar una **función por parámetro** a otra **función**, de modo que esta última función puede ejecutar la función pasada por parámetro de forma genérica desde su propio código, y nosotros podemos definirlas desde fuera de dicha función.

```
const action = function () {  
    console.log("Acción ejecutada.");  
};  
const mainFunction = function (callback) {  
    callback();  
};  
mainFunction(action);
```

- Definimos una función action que realiza una tarea.
- Definimos una función mainFunction, que recibe como parámetro una función callback genérica, que no sabemos que hace exactamente.
- Llamamos a mainFunction, pasándole como parámetro la función concreta, que en nuestro caso es action.

### HOF

Las **funciones de orden superior** o **HOF** (*High Order Functions*) son funciones que reciben por parámetro otra función y/o devuelven una función mediante el return.

Ésta es una primera aproximación para controlar la asincronía en Javascript.

Ejemplo: creamos un número decimal aleatorio entre 0 y 1. Hemos definido que al ejecutar doTask, exista un 50% de probabilidad de que ocurra un error (*valores entre 0 y 0.5*), o no ocurra error (*valores entre 0.5 y 1*):

```
const action = function () {  
    console.log("Acción ejecutada.");  
};  
const error = function () {  
    console.error("Ha ocurrido un error");  
};  
const doTask = function (callback, callbackError) {  
    const isError = Math.random() < 0.5;  
    if (!isError) callback();  
    else callbackError();  
};  
doTask(action, error);
```

### HOF

Podemos ejecutar la función `doTask()`, que es nuestra HOF, cambiando los callbacks según nos interese, sin necesidad de crear funciones con el mismo código repetido una y otra vez.

En el caso de que las funciones **callbacks** sean muy cortas, muchas veces utilizamos directamente la función anónima, sin necesidad de guardarla en una variable previamente:

```
const doTask = function (callback, callbackError) {  
    const isError = Math.random() < 0.5;  
    if (!isError) callback();  
    else callbackError();  
};
```

Se reescribiría:

```
doTask(function () {  
    console.log("Acción ejecutada.");  
},  
function () {  
    console.error("Ha ocurrido un error");  
});
```