# Sesión 6 – Semana 3

## Functions y Callbacks



Carrera 43 A # 34 - 155. Almacentro. Torre Norte. Oficina 701
Medellín (Antioquia), Colombia

# Contenido

1. Functions
    1. Definición
    2. ¿Cómo crear funciones?
    3. Tipos de funciones
        1. Funciones anónimas
        2. Funciones autoejecutables
        3. Arrow function
2. Callbaks

# Functions o Funciones

***¿Qué son?***

Es similar a un procedimiento, que es un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función, debe **tomar alguna entrada** y **devolver una salida** donde hay alguna **relación** obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamarla.

# Functions o Funciones

***¿Qué son?***

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

Con las funciones solucionamos:

1. Que el código de la aplicación sea mucho más largo porque muchas instrucciones están repetidas.
2. Si se quiere modificar alguna de las instrucciones repetidas, se deban hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que convierte la codificación en un trabajo muy pesado y muy propenso a cometer errores.

# ¿Cómo crear funciones?

Hay varias formas de crear funciones en JavaScript: por **declaración** (la más usada por principiantes), por **expresión** (la más habitual en programadores con experiencia) o mediante **constructor** de objeto (no recomendada):

| Constructor | Descripción |
| --- | --- |
| FUNCTION `function nombre(p1, p2...) { }` | Crea una función mediante **declaración**. |
| FUNCTION `var nombre = function(p1, p2...) { }` | Crea una función mediante **expresión**. |
| FUNCTION `new Function(p1, p2..., code);` | Crea una función mediante un constructor de **objeto**. |

# ¿Cómo crear funciones?

***Funciones por declaración***

Probablemente, la forma más popular de estas tres, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación, es la primera, a la **creación de funciones por declaración**. Esta forma permite declarar una función que existirá a lo largo de todo el código:

```
function saludar() {
    return 'Hola';
}

saludar();
console.log(typeof saludar);
```

MAKAIA

WWW.MAKAIA.ORG

# ¿Cómo crear funciones?

***Funciones por expresión***

Método donde se “guardan funciones” dentro de variables, para después “ejecutar dichas variables”:

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

saludo();
```

La diferencia fundamental entre las funciones por declaración y las funciones por expresión es que estas últimas 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 por expresión***

el nombre de la función (en este ejemplo: saludar) pasa a ser inútil, ya que si intentamos ejecutar `saludar()` nos dirá que no existe y si intentamos ejecutar `saludo()` funciona correctamente.

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

saludo();
```

**Nota**: La manera correcta de declarar este tipo de funciones es omitiendo el nombre de la función (“`saludar`” en este ejemplo) y dando paso a las **funciones anónimas** o **funciones lambdas**

MAKAIA

WWW.MAKAIA.ORG

# ¿Cómo crear funciones?

***Funciones como objetos***

Es posible, también, declarar funciones como si fueran objetos o a través de un constructor de un objeto.

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

saludar();
```



Carrera 43 A # 34 - 155. Almacentro. Torre Norte. Oficina 701
Medellín (Antioquia), Colombia

MAKAIA
WWW.MAKAIA.ORG

# Tipos de funciones

## *Funciones anónimas*

Las funciones anónimas o funciones lambda son un tipo de funciones que se declaran sin nombre de función y se alojan en el interior de una variable y haciendo referencia a ella cada vez que queramos utilizarla:

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

console.log(saludar); //[Function: saludar]
console.log(saludar()); //Hola
```

# Tipos de funciones

## *Funciones autoejecutables*

Estas funciones tienen como característica principal que una vez son declaradas, a continuación, se ejecutan.

```
//Función autoejecutable sin parámetros
(function () {
    console.log('Hola!!');
})();


//Función autoejecutable con parámetros
(function (name) {
    console.log(`Hola, ${name}`)
})('María');
```

En este caso lo que se almacena en la variable es el valor que devuelve la función autoejecutada.

```
const f = (function (name) {
    console.log(`Hola,
${name}`)
})('María');

f;
```

# Tipos de funciones

***Arrow functions o funciones flecha***

o «**fat arrow**» son una forma corta de escribir funciones que aparece en Javascript a partir de **ECMAScript 6**. Básicamente, se trata de reemplazar eliminar la palabra **`function`** y añadir **`=>`** antes de abrir las llaves:

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

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

MAKAIA

WWW.MAKAIA.ORG

# Tipos de funciones

***Arrow functions o funciones flecha***

Hacen que el código sea mucho más legible y claro de escribir, mejorando la productividad y la claridad a la hora de escribir código:

- Si el cuerpo de la función sólo tiene una línea, podemos omitir las llaves (`{}`).
- Además, en ese caso, automáticamente se hace un **`return`** de esa única línea, por lo que podemos omitir también el **`return`**.
- En el caso de que la función no tenga parámetros, se indica como en el ejemplo anterior: **`() =>.`**
- En el caso de que la función tenga un solo parámetro, se puede indicar simplemente el nombre del mismo: **`e =>.`**
- En el caso de que la función tenga 2 ó más parámetros, se indican entre paréntesis: **`(a, b) =>.`**
- Si queremos devolver un objeto, que coincide con la sintaxis de las llaves, se puede englobar con paréntesis: **`({name: 'Manz'}).`**

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

# Ámbito de **this**

Una de las principales diferencias de las funciones flecha respecto a las funciones tradicionales, es el valor de la palabra clave **this**, que no siempre es la misma.

```
// Si son funciones globales
const a = function () {
  console.log(this);
};
const b = () => {
  console.log(this);
};

a(); // Window
b(); // Window
```

Si se definen ambas funciones como variables globales **this** devuelve el objeto global **Window**

```
padre = {
  a: function () {
    console.log(this);
  },
  b: () => {
    console.log(this);
  },
};

padre.a(); // padre
padre.b(); // Window
```

Si se definen ambas funciones dentro de un objeto: la función tradicional devuelve **this** como el objeto **padre** de la función y la función flecha devuelve **Window**.

# Callbacks

Es una manera de ejecutar funciones dentro de otra función pasándolas por parámetro.

```
// fB = Función B
const fB = function () {
  console.log("Función B ejecutada.");
};

// fA = Función A
const fA = function (callback) {
  callback();
};

fA(fB);
```

# Ejemplos

1. Realizar una función que reciba 2 parámetros de tipo número, el segundo parámetro es opcional. Si se reciben los 2 parámetros debe retornar la suma de los 2 parámetros. Si se recibe un solo parámetro, debe retornar la suma de ese valor más 10.
2. Realizar una función que agregue elementos a un array previamente instanciado, y otra que quite elementos del array recibiendo un parámetro que será el indíce a eliminar. Si el índice no existe, debe eliminar el último elemento del array. Realizar una función que reciba alguna de las anteriores como parámetro callback y muestre el array restante.

# Ejercicios de práctica

1. Realizar una función que recibe 3 parámetros, los 2 primeros de tipo número, y el tercero un string indicando la operación a realizar: suma, resta, multiplicación o división. Debe retornar el resultado de la operación indicada.
2. Crea una función que genere 100 números aleatorios entre 1 y 1000 que no se repitan y luego muéstralos ordenados en consola.
3. Realizar una función que recibe 2 parámetros, el primero es un array de una lista de usuarios que debe contener nombre, apellidos y teléfonos, al menos 3 de esos usuarios deben comenzar con la letra “a”, el segundo parámetro es un callback. Debe procesar el array y eliminar todos los usuarios cuyo nombre inicie con la letra “a”. Posteriormente, enviar el nuevo array procesado al callback. El callback, debe mostrar en consola, la lista de todos los nombres, concatenando nombre y apellido.

# Fuentes


1. https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions
2. https://uniwebsidad.com/libros/javascript/capitulo-4/funciones
3. https://lenguajejs.com/javascript/fundamentos/funciones/

MAKAIA

WWW.MAKAIA.ORG

Info: comunicaciones@makaia.org

Corporación MAKAIA
Medellín, Colombia
Carrera 43A – 34-155. Almacentro
Torre Norte, Oficina 701
Teléfono: (+574) 448 03 74
Móvil: (+57) 320 761 01 76

@makaiaorg