



# Introducción a JavaScript 6-7-8-9 (ES6 o ES2015, ES7 o ES2016, ..)

# Índice

- Funciones
- Arrays

A horizontal yellow bar at the top of the slide, consisting of two segments of different lengths.

Funciones, array arguments, valores  
por defecto y operador spread

# Función

## Definición de la función

```
function my_preferred_movies () {  
  console.log();  
  console.log("My preferred movies:");  
  console.log(" - Jurassic Park by Steven Spielberg (1993)");  
  console.log(" - King Kong by Merian C. Cooper (1933)");  
  console.log(" - Citizen Kane by Orson Wells (1941)");  
  console.log();  
}
```

```
my_preferred_movies();
```

## Invocación (ejecución) de la función

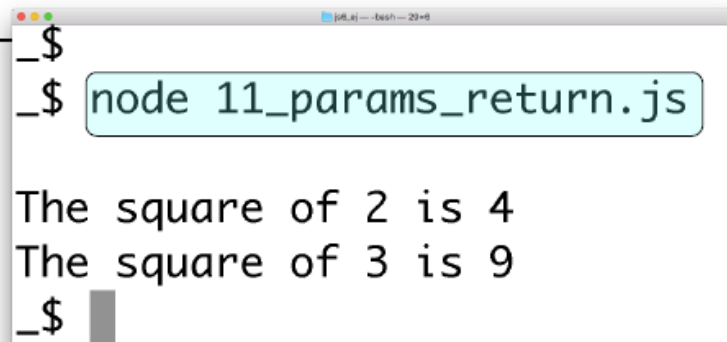
```
_ $  
_ $ node 10_function_movies.js  
  
My preferred movies:  
- Jurassic Park by Steven Spielberg (1993)  
- King Kong by Merian C. Cooper (1933)  
- Citizen Kane by Orson Wells (1941)  
  
_ $
```

Ejecución del programa  
10\_function\_movies.js  
con node.

- ◆ Una **función** encapsula código y lo representa por un **nombre**
  - Una función debe definirse primero, para poder invocarla (ejecutarla) posteriormente
    - ◆ Documentación: <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Funciones>
- ◆ La **definición** de la función comienza por la palabra reservada: **function**
  - A continuación viene el **nombre** de la función, que debe ser único en el programa
    - ◆ En tercer lugar vienen los parámetros entre paréntesis: ( ) indica sin parámetros en este ejemplo
      - Por último viene el bloque de código, entre corchetes { }
- ◆ La **invocación** de la función ejecuta el bloque de código de la función
  - Se invoca con el nombre y el operador paréntesis ( ), por ej. **my\_preferred\_movies()**

# Parámetros de invocación y de retorno

```
function square (x) {  
  return x*x ;  
}  
  
console.log();  
console.log("The square of " + 2 + " is " + square(2));  
console.log("The square of " + 3 + " is " + square(3));
```



A terminal window titled 'js:11 --- bash --- 20x8' showing the command 'node 11\_params\_return.js' being executed. The output of the command is 'The square of 2 is 4' followed by 'The square of 3 is 9'. The prompt is '\_\$'.

```
_$_  
_ $ node 11_params_return.js  
  
The square of 2 is 4  
The square of 3 is 9  
_ $
```

- ◆ Una **función** recibe parámetros de entrada (parámetro **x** del ejemplo)
  - Y devuelve un valor con la sentencia: **return <expr>**
    - ◆ Esta sentencia finaliza la ejecución de la función y devuelve el valor resultante de evaluar **<expr>**
  - Si la **función** llega a final del bloque **sin ejecutar** return, finaliza y devuelve **undefined**
- ◆ Un **parámetro** de una función es similar a la definición de una **variable**
  - El parámetro solo es **visible** dentro del **bloque de la función**
    - ◆ El parámetro se **inicia** con el **valor pasado al invocar la función**, en el ejemplo con los valores 2 y 3
- ◆ Una función puede usarse en expresiones como otro valor más
  - La función se ejecutará y se sustituirá por el valor devuelto en la expresión
    - ◆ En el ejemplo (return **x\*x** ;) devuelve el **cuadrado** del valor pasado en el parámetro **x**

# Número de parámetros de una función

Antes de ES6 los strings se concatenaban así:  
**greeting + " " + person + ", how are you?"**  
(es prácticamente equivalente, pero menos compacto)

```
function greet (greeting, person) {  
  return `${greeting} ${person}, how are you?`;  
};
```

```
greet ("Good morning", "Peter");    // => "Good morning Peter, how are you?"
```

```
greet ("Hi", "Peter");              // => "Hi Peter, how are you?"
```

```
greet ("Hi", "Peter", "Bill");      // => "Hi Peter, how are you?"
```

```
greet ("Hi");                      // => "Hi undefined, how are you?"
```

```
greet ();                          // => "undefined undefined, how are you?"
```

## ◆ Una función **se puede invocar** con un **número variable de parámetros**

- Un parámetro definido, pero **no pasado** en la invocación, toma el valor **undefined**
  - ◆ Un parámetro pasado en la invocación, pero **no utilizado**, no tiene utilidad

## ◆ La función **greet(..)** genera un saludo utilizando 2 parámetros

- El ejemplo ilustra como procesa JavaScript parámetros no pasados o no utilizados

# arguments: el array con los parámetros

```
function greet () {  
    return `${arguments[0]} ${arguments[1]}, how are you?`;  
};  
  
greet ("Good morning", "Peter");    // => "Good morning Peter, how are you?"  
  
greet ("Hello", "Peter");           // => "Hello Peter, how are you?"
```

- ◆ Una función tiene predefinida un array de nombre **arguments**
  - **arguments** contiene los valores asignados a los parámetros en la invocación
    - ◆ Aquí se define la función **greet** utilizando **arguments** en vez de parámetros explícitos
      - Por último viene el bloque de código, entre corchetes {...}
- ◆ Una función se puede invocar con un número variable de parámetros
  - El array **arguments** permite saber su número y acceder a todos



# Resto de parametros en ES6: ...x

- ◆ Operador spread (...x) da acceso al resto de los parámetros de una función en ES6
  - Los parámetros están accesibles a través del array asociado al operador
    - ♦ Parámetros explícitos y operador rest pueden mezclarse entre sí, por ejemplo: `function f1 (x, y, ...resto) {...}`
      - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest\\_parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters)
      - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\\_operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator)
- ◆ los ejemplos muestran 2 definiciones equivalentes de la función greet

```
function greet (...args) {  
  return `${args[0]} ${args[1]}, how are you?`;  
};
```

```
greet ("Good morning", "Peter");    // => "Good morning Peter, how are you?"  
greet ("Hello", "Peter");           // => "Hello Peter, how are you?"
```

```
function greet (greeting, ...more) {  
  return `${greeting} ${more[0]}, how are you?`;  
};
```


```
greet ("Good morning", "Peter");    // => "Good morning Peter, how are you?"  
greet ("Hello", "Peter");           // => "Hello Peter, how are you?"
```



# Valores por defecto de parámetros (ES6)

```
function greet (greeting = "Hi", person = "my friend") {  
    return `${greeting} ${person}, how are you?` ;  
};  
  
greet ("Hello");           // => "Hello my friend, how are you?"  
greet ();                  // => "Hi my friend, how are you?"
```

- ◆ **ES6** permite asignar valores por defecto a parámetros de funciones
  - Los valores por defecto se asignan al parámetro en la definición
    - ◆ utilizando el operador =, como en las definiciones de variables
- ◆ El valor por defecto se utiliza si la invocación no incluye ese parámetro

A horizontal decorative bar at the top of the slide, consisting of a small yellow square on the left and a longer yellow rectangle extending to the right.

Arrays, spread y métodos sort,  
reverse, concat, join, indexOf, slice,  
splice, push y pop

# Arrays

## ◆ Array

- Es una colección ordenada de elementos
  - ◆ Se suele crear con el literal de array: [7, 4, 2, 23]
    - El operador corchetes agrupa elementos en arrays
- **toString()** devuelve un string con los elementos

## ◆ Los elementos de un array de tamaño n

- se acceden con un índice entre 0 y n-1
  - ◆ **a[k]** accede al elemento k+1

## ◆ **a.length** indica el tamaño del array

- Un array tiene un máximo de  $2^{32}-2$  elementos

## ◆ Cambiar **length** cambia el tamaño del array

- Por ejemplo, **a.length = 2** reduce el tamaño de **a** a 2
  - ◆ Quedando solo los dos primeros elementos

## ◆ El **operador spread (...x)** nuevo en ES6

- Inserta los elementos de un array en otro array

```
let a = [7, 4, 1, 23];
```

```
a[0]      => 7
```

```
a[1]      => 4
```

```
a[2]      => 1
```

```
a[3]      => 23
```

```
a.toString() => "7,4,1,23"
```

```
a.length  => 4
```

```
let a = [7, 4, 1, 23];
```

```
a.length = 2    => 2
```

```
a              => [7, 4]
```

```
let a = [7, 4, 1];
```

```
let b = [0, 0, ...a];
```

```
b              => [0, 0, 7, 4, 1]
```

```
b.length      => 5
```

# Métodos para ordenar, invertir, concatenar o buscar

## ◆ **sort()**

Estos métodos no modifican el array original, solo devuelven el resultado como parámetro retorno.

- devuelve el array ordenado

```
[1, 5, 3].sort() // => [1, 3, 5]
```

## ◆ **reverse()**

- devuelve el array invertido

```
[1, 5, 3].reverse() // => [3, 5, 1]
```

## ◆ **concat(e1, ..., en)**

- devuelve un nuevo array con **e1, ..., en** añadidos al final

```
[1, 5, 3].concat(9) // => [1, 5, 3, 9]  
[1, 5, 3].concat(9, 3) // => [1, 5, 3, 9, 3]
```

## ◆ **join(<separador>)**

- concatena elementos en un string
  - ◆ introduce <separador> entre elementos

```
[1, 5, 3, 7].join(';') // => '1;5;3;7'  
[1, 5, 3, 7].join('') // => '1537'
```

## ◆ **indexOf(elem, offset)**

- devuelve índice de primer **elem**
  - ◆ **offset**: comienza búsqueda (por defecto 0)

```
[1, 5, 3, 5, 7].indexOf(5) // => 1  
[1, 5, 3, 5, 7].indexOf(5, 2) // => 3
```

```
[1, 5, 3].concat(2).sort().reverse() // => [5, 3, 2, 1]
```

Los métodos encadenados aplican el segundo método sobre retorno del primero.

# Extraer, modificar o añadir elementos al array

◆ **slice(i,j):** devuelve la rodaja entre **i** y **j**

- Índice negativo (**j**) es relativo al final
  - ◆ índice "-1" es igual a `a.length-2`
- No modifica el array original

◆ **splice(i, j, e1, e2, ..., en)**

- sustituye **j** elementos desde **i** en array
  - ◆ por **e1, e2, ...,en**
- Devuelve rodaja eliminada

◆ **push(e1, ..., en)**

- añade **e1, ..., en** al final del array
  - ◆ devuelve el tamaño del array (`a.length`)

◆ **pop()**

- elimina último elemento y lo devuelve

```
[1, 5, 3, 7].slice(1, 2) => [5]
[1, 5, 3, 7].slice(1, 3) => [5, 3]
[1, 5, 3, 7].slice(1, -1) => [5, 3]
```

```
let a = [1, 5, 3, 7];
```

```
a.splice(1, 2, 9) => [5, 3]
a                  => [1, 9, 7]
```

```
a.splice(1,0,4,6) => []
a                  => [1, 4, 6, 9, 7]
```

```
let b = [1, 5, 3];
```

```
b.push(6, 7)      => 5
b                  => [1, 5, 3, 6, 7]
```

```
b.pop()           => 7
b                  => [1, 5, 3, 6]
```