Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Introducción a JavaScript I

IIC2513 - Tecnologías y Aplicaciones Web

Sebastián Vicencio R.
2do Semestre 2020

JavaScript

Lenguaje de programación
que permite desarrollar
aplicaciones web modernas

Creado en 1995

Brendan Eich

Netscape Communications Corporation

JavaScript !== Java

Al igual que casilla es distinto a silla

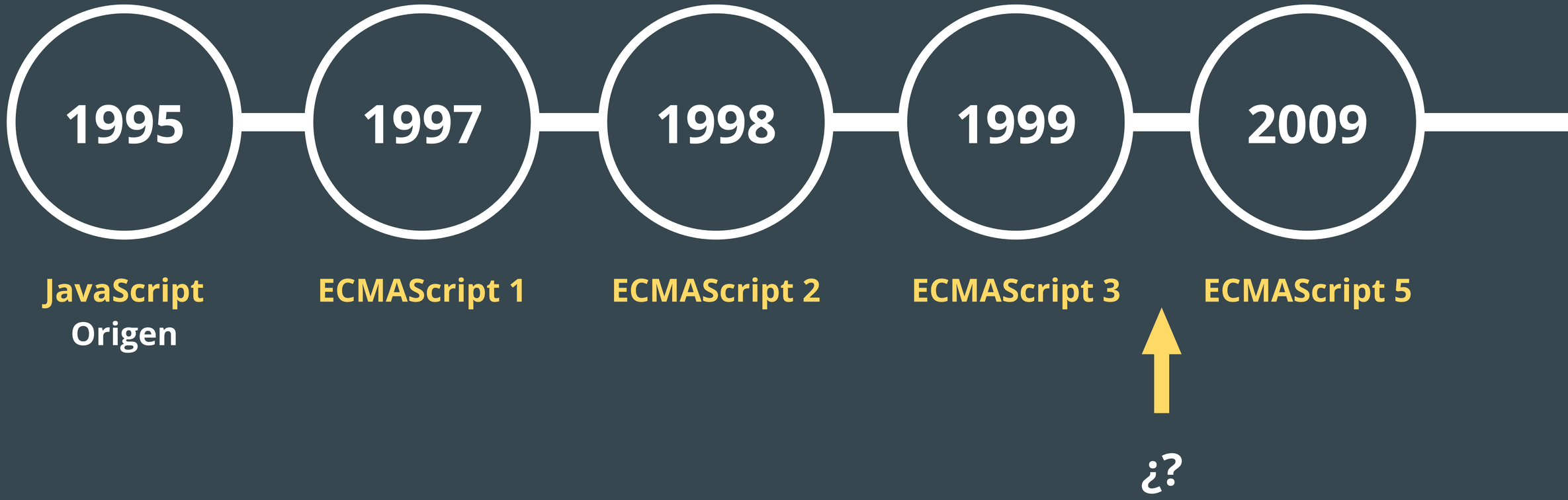
ECMAScript

Estándar por ECMA International

- Comité TC39
- Define evolución del lenguaje

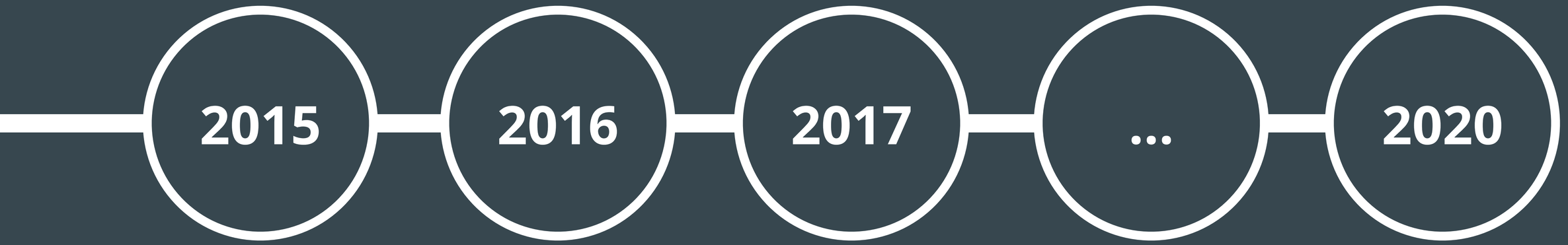
Evolución JavaScript (ES)

Versiones



Evolución JavaScript (ES)

Versiones



ECMAScript 6
ES6 o ES2015

ES2016

ES2017

Versión anual
Features con stages

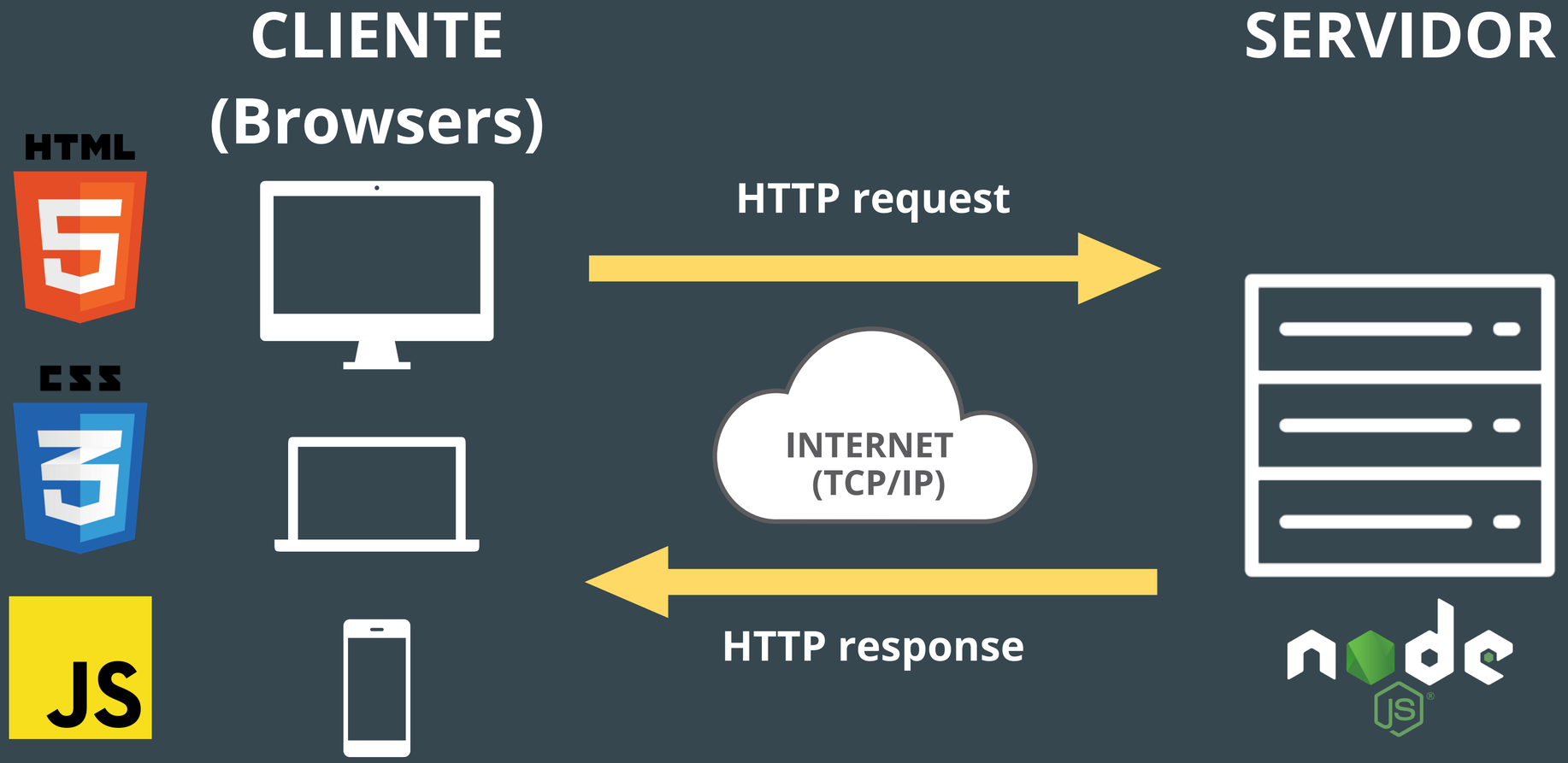
ES2020

Más información: https://github.com/tc39/ecma262

¿Dónde se sitúa JavaScript?

Ambiente de ejecución

¿Dónde se sitúa JavaScript?



Motores JavaScript

Ejecución del lenguaje

V8
Chrome / Node.js

SpiderMonkey
Firefox

JavaScriptCore
Safari

Chakra
Edge

Hello World

Console

```
console.log("Hello World”);
```

- Debuggear código en ambiente de ejecución
- Imprimir output

JavaScript

Características básicas

Características básicas

Interpretado

```
let myVariable = 5;

myVariable = ‘hello';
```

Dinámico

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

console.log(add(3, 5));
console.log(add('Ho', ‘la’));
```

Weakly-typed

```
console.log(3 + '–tree');
```

Características básicas

Multi-paradigma

Object-oriented

```
let myVehicle = new Vehicle();
myVehicle.drive();
```

Imperativo

```
let result = '';
for (let i = 0; i <= 20; i += 1) {
  let isEven = i % 2 === 0;
  if (isEven) result += `${i}\n`;
}
console.log(result);
```

Funcional

```
let result = Array
  .from(Array(21), (x, i) => i)
  .filter(x => x % 2 === 0)
  .join('\n');
console.log(result);
```

Características básicas

Object-oriented con prototypes

- Nuevos objetos a partir de prototipos
- Originalmente sin clases
- Desde ES6 clases como syntactic sugar

```
let Vehicle = function() {}

Vehicle.prototype.drive = function() {
  console.log('I\'m driving');
}
```

Características básicas

Duck typing

If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.

```
duck.fly();
// Flying!

eagle.fly();
// Flying!

dog.fly();
// TypeError: dog.fly is not a function
```

Características básicas

Single-threaded

No existen threads

Sólo una operación en un determinado momento

Asynchronous

Permite ejecutar código luego de que ocurra cierto evento

Sin interrumpir la ejecución principal

JavaScript

Elementos fundamentales

Valores

Representación de información

```
2
"hola"
undefined
{}
true
```

Valores

Tipos de valores

Valores primitivos

Inmutables

`Number`

`String`

`Boolean`

`undefined`

`null`

`BigInt`

`Symbol`

Valores no primitivos

Mutables

`Object`

`Function`

Valores

Number

- 2^{64} valores posibles
- Incluye números negativos y decimales

```
2

0.1

-1000000000
```

Valores

String

- Representan texto
- Single o double quotes
- Escapar caracteres: \char

```
“hola"
```

```
‘curso'
```

```
"Una línea\nOtra línea"
```

Valores

`Boolean`

Dos valores

`true` `false`

`undefined`

Un valor
Valor ausente no intencional

`null`

Un valor
Valor ausente intencional

Expresiones

Una **pregunta** que hacemos

2 + 3

Siempre **produce un valor**

Statements

```
expression;
```

Primer tipo de statement

Un programa es una lista de statements

Variables

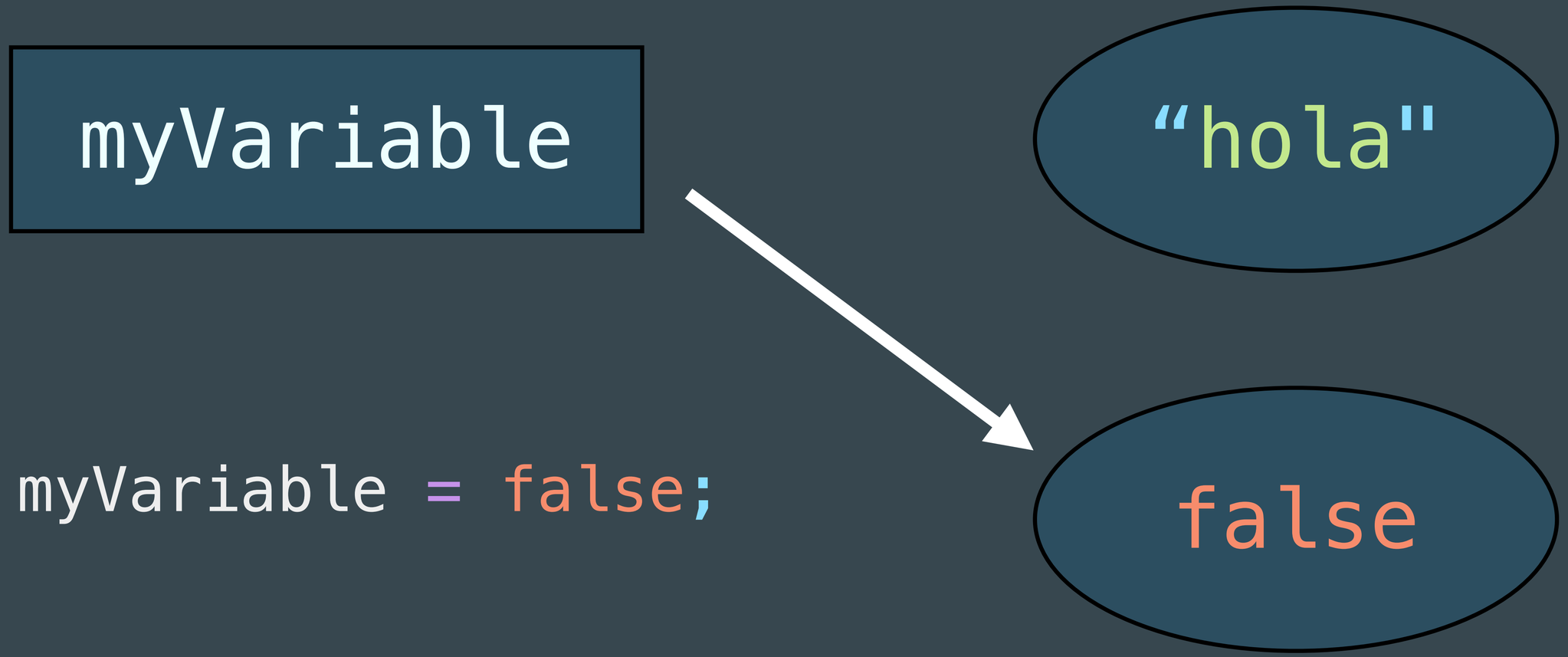
Apuntan a un valor


```
let myVariable = expression;
```

Segundo tipo de statement

Variables

Re-asignación



Variables

Keywords

var

```
a = 5;

var a;
```

Hoisted

let

```
b = 5;

let b;
// ReferenceError: can’t
// access lexical
// declaration ‘b’
// before initialization
```

const

```
const c = 5;

c = 10;
// TypeError: invalid
// assignment to const 'c'
```

Operaciones

Aritméticas y asignación

Números

`+` `-` `*` `/` `%`

Concatenación strings

`+`

Asignación

`+=`

`-=`

`*=`

`/=`

Operaciones

Comparación

```
>  <  >=  <=
```

Strict equality

```
===  !==
```

Type coercion

```
5 == '5' // true

'' == 0 // true

false == 0 // true
```

Operaciones

Lógicas

`&&` `||` `!`

Short circuiting

```
expression1 && expression2 // Guard

expression1 || expression2 // Default
```

Condicionales

Control de flujo

Condicionales

```
if (condition) {
  // Code to run if condition is met
}
```

Block { }

Tercer tipo de statement

Condicionales

```
if (condition) {
  // Code to run if condition is met
} else {
  // Code to run otherwise
}
```

```
if (condition1) {
  // Code to run if condition1 is met
} else if (condition2) {
  // Code to run if condition1 is not met but condition2 is
} else {
  // Code to run otherwise
}
```

Condicionales

```
let myVariable = condition ? expression1 : expression2;
// If condition is met, expression1 is evaluated
// If not, expression2 is evaluated
```

Ternary operator

Loops

Control de flujo

Loops

while

```
while (condition) {
  // Statements
}
```

Condición de término

¡No olvidar!

```
let i = 0;
while (i < 20) {
  // Statements
  i += 1;
}
```

Loops

for

```
for (init; condition; update) {
  // Statements
}
```

Más conveniente

```
for (let i = 0; i < 20; i += 1) {
  // Statements
}
```

Loops

Manejo de iteraciones

```
break;
```

Termina ejecución loop

```
continue;
```

Pasa a siguiente iteración

Funciones

Encapsular código

Funciones

Valor no primitivo

- Encapsular trozo código en un valor
- Evitar repetir código
- Se “llama” para ejecutar código

```
function fnName(param1, param2, ..., paramN) {
  // Statements (body)
}
```

Funciones

Ejecución

```
fnName('This is a parameter')
```

La llamada es una expresión

```
return expression;
```

Valor de retorno

Actividad

Hands-on