JavaScript I

Semana 1

MigraCode

Alexandra Yamaui

Qué es JavaScript?

- HTML
- CSS (estilos)

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class='item', id='div-tag'>
                  This is an item
            </div>
            <a href='https://migracode.eu/'>
                  Link to Migracode website
            </a>

      </body>
</html>
```

Qué es JavaScript?

Javascript

- Lenguaje para manipulación de páginas web (HTML)
- Añadir elementos HTML dinámicamente
- Cambiar el contenido HTML actual
- Modificar estilos (CSS)
- Reaccionar a las interacciones del usuario, como el click del mouse, movimientos del puntero, presionar una tecla.

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class='item', id='div-tag'>
                  This is an item
            </div>
            <a href='https://migracode.eu/'>
                  Link to Migracode website
            </a>

      </body>
</html>
```

Qué es JavaScript?

Javascript

- Enviar requests a servidores remotos, descargar y subir archivos.
- Leer y crear cookies.
- Almacenar información en el browser (“local storage”).

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class=’item’, id=’div-tag’>
                  This is an item
            </div>
            <a href=’https://migracode.eu/’>
                  Link to Migracode website
            </a>

      </body>
</html>
```

Cómo funciona JavaScript?

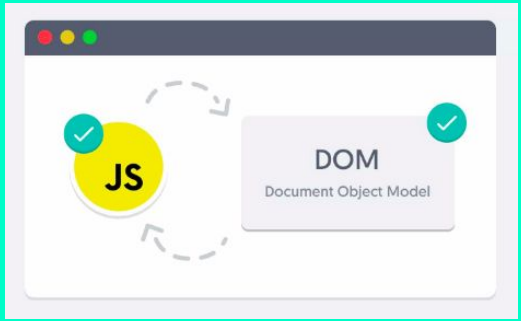
El Browser tiene un **motor de JavaScript** integrado

- Programa que ejecuta código JavaScript
- Se ejecuta a la par del **motor de renderizado** a través del Document Object Model (**DOM**).

Cómo funciona JavaScript?

DOM (Document Object Model)

- Representación de un documento HTML en el navegador
- El DOM es la manera en que JavaScript interactúa con el HTML y los estilos (CSS)



Cómo funciona JavaScript?

Insertamos código JavaScript dentro del document HTML delimitado por las etiquetas **<script></script>**

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
            <script>
                  Javascript code
            </script>
      </head>
      <body>

            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
            <div class='item', id='div-tag'>
                  This is an item
            </div>
            <a href='https://migracode.eu/'>
                  Link to Migracode website
            </a>

      </body>
</html>
```

Cómo funciona JavaScript?

Insertamos código JavaScript dentro del document HTML delimitado por las etiquetas **<script></script>**

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
            <script>
                  console.log("Hello World!");
                  console.log("I'm Learning
            JavaScript...");
                  console.log("in Migracode");
            </script>
      </head>
      <body>
            ...
      </body>
</html>
```

Cómo funciona JavaScript?

En vez de insertar el código JavaScript directamente en el HTML, podemos escribirlo en un archivo JavaScript (con extensión **.js**) y hacer referencia a él en el HTML

```
<!DOCTYPE html>
<html>
      <head>
            <title>Page Title</title>
            <script src='example.js'></script>
      </head>
      <body>
            …
      </body>
</html>
```

Cómo funciona JavaScript?

Script:

- Secuencia de instrucciones
- Las instrucciones se ejecutan de una en una desde el inicio del archivo hasta el final. El **orden** de ejecución se llama hilo de ejecución.
- Cada instrucción se delimita con un punto y coma (;)

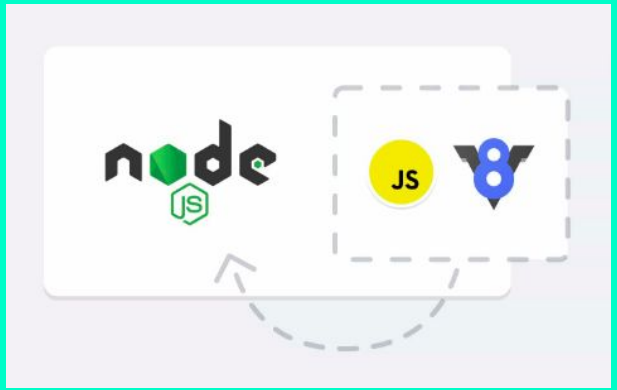
example.js

```
console.log("Hello World!");                    1
console.log("I'm Learning JavaScript...");      2
console.log("in Migracode");                    3
```

Qué es Node JS?

Node JS

- Procesador/Motor de JavaScript fuera del navegador



Ejercicios

Ejercicio A (5 minutos)

(Este ejercicio te ayudará a entender cómo ejecutar un script básico de JavaScript y explorar la diferentes maneras de ejecutar un código JS)

1. Modifica el script `exercise-A.js` en el directorio `week-1/InClass` en el repositorio Javascript/Core Module 1 al que le hiciste *fork* anteriormentte
2. Escribe `console.log("Hello World!")` en el script
3. Ejecuta el script yendo desde el terminal al directorio donde está el archivo y escribiendo en el terminal **`node exercise-A.js`**, verás el resultado del console.log en el terminal

Ejercicios

Ejercicio B (5 minutos)

(Este ejercicio te ayudará a mejorar tu entendimiento de console.log)

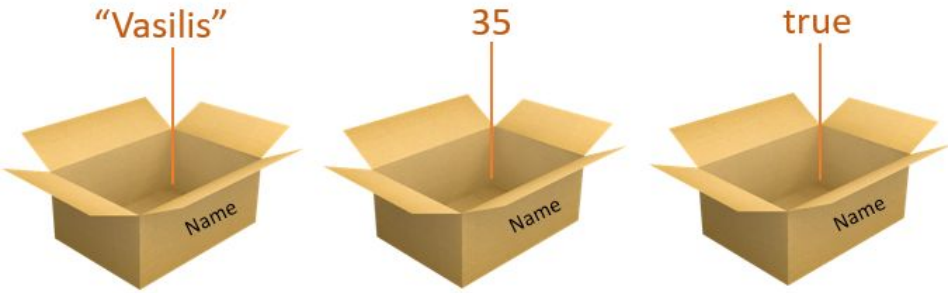
1. Modifica el script `exercise-B.js` en el directorio `week-1/InClass`
2. Escribe 5 mensajes usando console.log, pero en diferentes idiomas.

Ejemplo del resultado:

```
Halo, dunia! // Indonesian
Ciao, mondo! // Italian
Hola, mundo! // Spanish
```

JS Variables

Contenedores para guardar valores



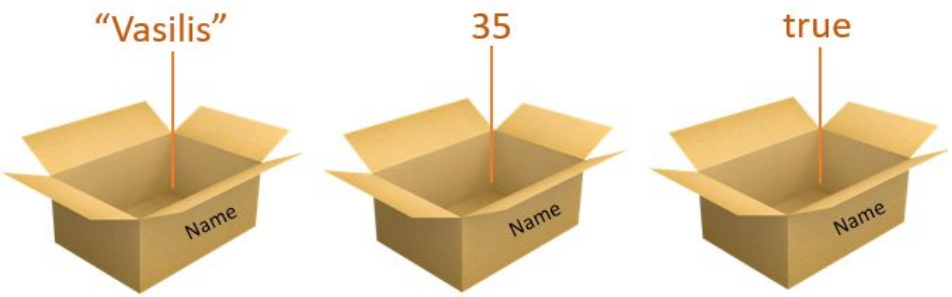
example.js

```
let variable1 = "Vasilis";

console.log(variable1);

variable1 = 35;
variable1 = true;
```

JS Variables

Hay tres **claves (palabras reservadas)** para crear variables:

- const
- let
- var

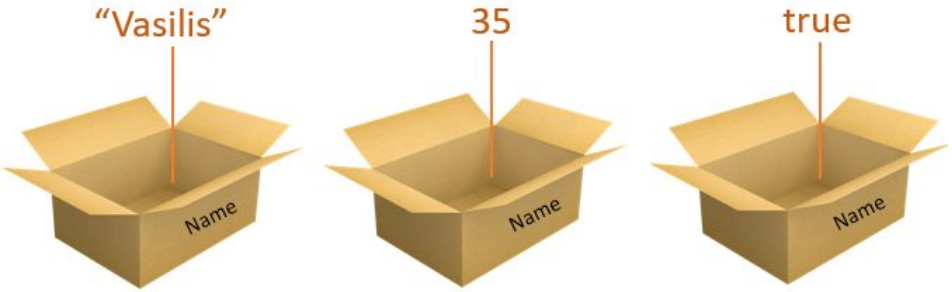


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **const**
  - Su valor no se puede cambiar
  - Si se trata de reemplazar su valor, saltará un Error

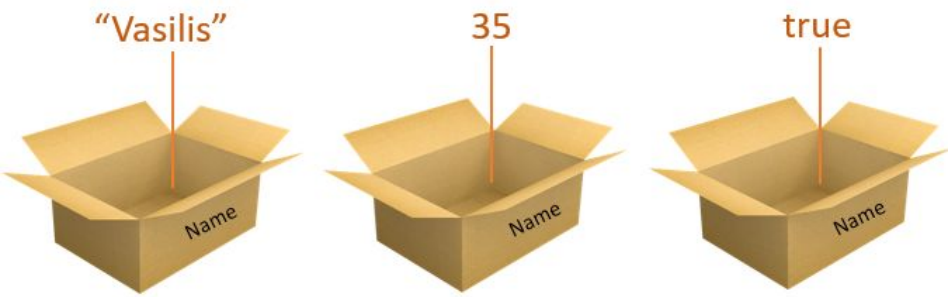


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **let**
  - Su valor se puede reemplazar con un nuevo valor

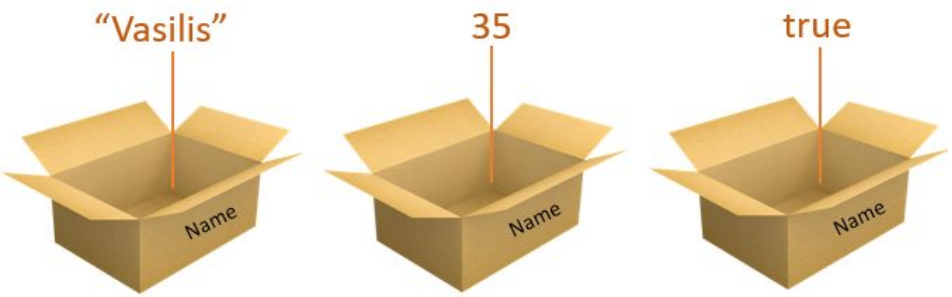


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **var**
  - Su valor se puede reemplazar con un nuevo valor

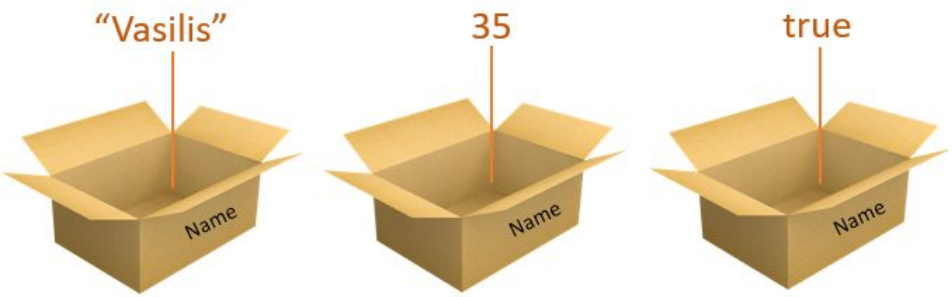


example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

JS Variables

- **let vs var**
  - Tienen diferentes **alcances** (lo veremos más adelante)
  - Es preferible usar **let** para evitar errores inesperados



example.js

```
const constantVariable = "constant value";

console.log(constantVariable);


let rewritableVariable = "dynamic value";

console.log(rewritableVariable);


var rewritableVariable = "dynamic value";

console.log(rewritableVariable);
```

Ejercicios

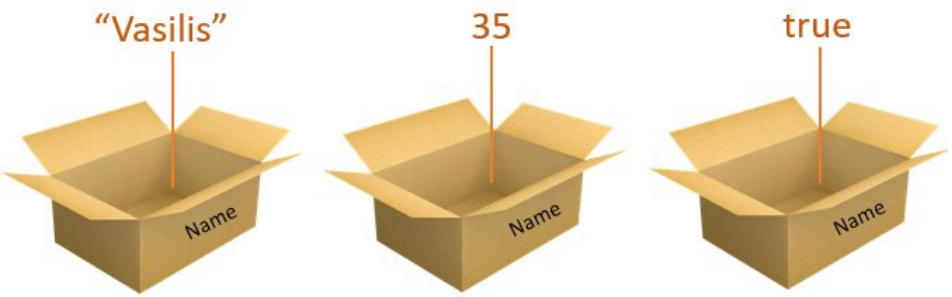
Ejercicio C (5 minutos)

1. Modifica el script `exercise-C.js` en el directorio `week-1/InClass`
2. Crea una variable **`greeting`** y guarda en la variable un saludo de tu escogencia
3. Imprime en la cónsola tu saludo 3 veces. Deberías ver tu saludo en el terminal 3 veces, cada una en una línea distinta.

JS Tipos de datos

- Number
- String
- Float
- Boolean

- Function
- Array
- Object
- Undefined
- Null



example.js

```
let name = "Vasilis"; // String type  <- Comment

console.log(name);

let name = 35; // Number type

console.log(name);

let name = true; // Boolean type

console.log(name);
```

JS Tipos de datos

- Para saber el tipo de una variable podemos usar el operador typeof

example.js

```
const message = "This is a string";

const messageType = typeof message;

console.log(messageType); // logs 'string'
```

JS Strings

- Cadena (string) de caracteres
- Se definen usando comillas (pueden ser comillas simples (‘) or dobles (“))

example.js

```
const message = "This is a string";
```

Ejercicios

Ejercicio D (10 minutos)

1. Modifica el script `exercise-D.js` en el directorio `week-1/InClass`
2. Escribe un programa que:
    - Cree una variable llamada **`colors`**
    - Asigne los colores "blue" y yellow" separados por coma a la variable **`colors`**
    - Imprima el tipo de la variable **`colors`** usando el operador **`typeof`**.
3. Cual es el resultado de aplicar el operador **`typeof`** a un número?

JS Strings

Concatenación de strings:

- Dos strings pueden ser concatenados usando el operador de suma (+)

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";

const greeting = greetingStart + name;

console.log(greeting); // Logs "Hello, my name
Alexandra"
```

JS Strings

Concatenación (interpolación) de strings:

- Podemos insertar (interpolar) valores en los strings dinámicamente escribiendo el texto entre **backticks** (``) y envolviendo las **variables** que queremos insertar con ${}

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";

const greeting = `${greetingStart} ${name}`;

console.log(greeting); // Logs "Hello, my name is Alexandra"
```

Ejercicios

Ejercicio E (5 minutos)

1. Modifica el script `exercise-E.js` en el directorio `week-1/InClass`
2. Escribe un programa que imprima un mensaje con un saludo y tu nombre usando los dos métodos de concatenación que vimos

JS Numbers

- No se escriben dentro de comillas
- Representan números enteros

Operadores:

- Suma (+)
- Resta (-)
- División (/)
- Multiplicación (*)

example.js

```
const age = 30;

const sum = 10 + 2; // 12
const product = 10 * 2; // 20
const division = 10 / 2; // 5
const difference = 10 - 2; // 8
```

Ejercicios

Ejercicio F (10 minutos)

1. Modifica el script `exercise-F.js` en el directorio `week-1/InClass`
2. Crea dos variables **`numberOfStudents`** y **`numberOfMentors`**
3. Imprime tres mensajes que muestren el número de estudiantes, el número de mentores y el total de estudiantes y mentores

Ejemplo resultado

```
Number of students: 15
Number of mentors: 8
Total number of students and mentors: 23
```

JS Floats

- Representan números con decimales (números punto flotante)

Operadores:

- Suma (+)
- Resta (-)
- División (/)
- Multiplicación (*)

example.js

```
const preciseAge = 30.612437;
```

JS Floats

- Redondeo de floats

example.js

```
const preciseAge = 30.612437;
const roughAge = Math.round(preciseAge); // 31
```

JS Librerías

- Código JavaScript escrito por alguien más que podemos usar en nuestros scripts
- Algunas librerías están pre-cargadas automáticamente
- Pueden descargarse de internet
- Ex: La librería **Math** ya está pre-cargada y podemos usarla sin hacer nada

example.js

```
const preciseAge = 30.612437;
const roughAge = Math.round(preciseAge); // 31
```

Ejercicios

Ejercicio G (15 mins)

1. Modifica el script `exercise-G.js` en el directorio `week-1/InClass`
2. Usando las variables creadas en el ejercicio F, calcular el porcentaje de mentores y estudiantes en el grupo (los porcentajes deben ser redondeados al entero más cercano)
3. Usando documentación en internet, di qué otras cosas puedes hacer con la librería `Math`? Escoge una método diferente a `Math.round` y explícalo al resto de la clase

Ejemplo resultado

```
Percentage students: 65%
Percentage mentors: 35%
```

JS declaración vs inicialización de variables

Para asignar un valor a una variable, primero necesitamos **declararla** y luego **inicializarla** con un valor. Puede hacerse la declaración e inicialización en la misma línea.

example.js

```
// Variable declaration
let x;

// Variable initialization
x = 2;

// Variable declaration and initialization
let x = 2;
```

JS Funciones

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Funciones

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Daniel";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Ana";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Funciones

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Daniel";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Ana";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Funciones

Función:

- Bloque **reusable** de instrucciones diseñado para realizar una tarea específica
- Primero necesitamos **definir** la función y luego **invocarla** (**llamarla**)
- Ex: console.log() es una función que viene disponible al cargar JavaScript

example.js

```
const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Daniel";
const greeting = greetingStart + name;
console.log(greeting);

const name = "Ana";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función (si los hay)

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}
```

JS Funciones

Bloque de código:

Un bloque define un conjunto de instrucciones que se ejecutan juntas de manera secuencial

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
  // code to be executed
}
```

Block

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);


function greetingFcn(...) {




}
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

const greetingStart = "Hello, my name is ";
const name = "Alexandra";
const greeting = greetingStart + name;
console.log(greeting);


function greetingFcn(...) {
 const greetingStart = "Hello, my name is ";
 const name = "Alexandra";
 const greeting = greetingStart + name;
 console.log(greeting);
}
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

function greetingFcn(...) {
 const greetingStart = "Hello, my name is ";
 const name = "Alexandra";
 const greeting = greetingStart + name;
 console.log(greeting);
}
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

function greetingFcn(...) {
 const greetingStart = "Hello, my name is ";
 const name = "Alexandra";
 const greeting = greetingStart + name;
 console.log(greeting);
}

function greetingFcn(name) {
 const greetingStart = "Hello, my name is ";
 const name = "Alexandra";
 const greeting = greetingStart + name;
 console.log(greeting);
}
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

function greetingFcn(name) {
 const greetingStart = "Hello, my name is ";
 const greeting = greetingStart + name;
 console.log(greeting);
}


// How can we make the greeting also configurable?
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

function greetingFcn(name) {
 const greetingStart = "Hello, my name is ";
 const greeting = greetingStart + name;
 console.log(greeting);
}
```

JS Funciones

Definición de funciones:

- Una función se define usando la palabra clave function, seguido del nombre de la función y paréntesis
- Dentro los paréntesis se escriben los parámetros de la función

example.js

```
function functionName(parameter1, parameter2,
parameter3, ...) {
 // code to be executed
}

function greetingFcn(name, greetingStart) {
 const greetingStart = "Hello, my name is ";
 const greeting = greetingStart + name;
 console.log(greeting);
}
```

JS Funciones

Invocación de funciones:

- Para invocar una función necesitamos escribir el nombre de la función seguido de paréntesis ()
- El orden de los parámetros es importante

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 console.log(greeting);
}

// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
```

JS Funciones

Invocación de funciones:

- Para invocar una función necesitamos escribir el nombre de la función seguido de paréntesis ()
- El orden de los parámetros es importante

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 console.log(greeting);
}

// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
greetingFcn("Daniel", "Hello, I'm ");
greetingFcn("Ana", "Hi!, my name is ");
```

JS Funciones

Valor de retorno:

- Una función puede retornar un valor o no
- Para retornar un valor usamos la palabra clave return
- Cuando la ejecución del programa llega a línea de return automáticamente sale del bloque de la función y vuelve al programa principal
- Tenemos que hacer algo con el valor de retorno

example.js

```
// Function definition
// This function does not have a return value,
// it only logs a message in the terminal
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 console.log(greeting);
}


// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");
```

JS Funciones

El valor retornado por una función se puede guardar en una variable

example.js

```
// Function definition
// This function has a return value (greeting)
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 return greeting;
}

// Function invocation
const resultGreeting = greetingFcn("Alexandra",
"Hello, my name is ");

console.log(resultGreeting);
```

JS Funciones

El valor retornado por una función se puede guardar en una variable

example.js

```
// Function definition
// This function has a return value (greeting)
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 return greeting;
}

// Function invocation
const resultGreeting = greetingFcn("Alexandra",
"Hello, my name is ");

console.log(resultGreeting);
```

JS Funciones

El valor retornado por una función se puede guardar en una variable

example.js

```
// Function definition
// This function has a return value (greeting)
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 return greeting;
}

// Function invocation
const resultGreeting = "Hello, my name is Alexandra"

console.log(resultGreeting);
```

JS Funciones

El valor retornado por una función se puede guardar en una variable

example.js

```
// Function definition
// This function has a return value (greeting)
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 return greeting;
}

// Function invocation
const resultGreeting = "Hello, my name is Alexandra"

console.log(resultGreeting);

console.log(greetingFcn("Hello, my name is ",
"Alexandra")); // ?
```

Ejercicios

Ejercicio H (20 minutos)

1. Modificar el script `exercise-H.js` en el directorio `week-1/InClass`
2. Crear una función que:
    i. Reciba más de un parámetro
    ii. Use concatenación de strings (unir dos strings)
    iii. Realice una operación matemática
    iv. Retorne un string
3. Agregar un comentario arriba de la definición de la función explicando qué hace
4. Llamar la función y ejecutar el script
5. Cual es la diferencia entre **`return`** y **`console.log`**?
6. Cuándo es útil utilizar funciones?

JS Funciones

Las funciones alteran el hilo de ejecución del programa (script).

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;
 return greeting;
}



console.log("First print");

// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");

console.log("Last print");
```

JS Funciones

Las funciones alteran el hilo de ejecución del programa (script).

example.js

```
// Function definition
function greetingFcn(name, greetingStart) {
 const greeting = greetingStart + name;          3
 return greeting;                                4
}



console.log("First print");                      1

// Function invocation
greetingFcn("Alexandra", "Hello, my name is ");  2

console.log("Last print");                       5
```

JS Funciones

Podemos llamar (invocar) funciones dentro de otras funciones.

example.js

```
// Function definition
function getAgeInDays(age) {
 return age * 365;
}

// Function definition
function createGreeting(name, age) {
 const ageInDays = getAgeInDays(age);
 const message = "My Name is " + name + " and I
was born over " + ageInDays + " days ago!";
 return message;
}
```

JS Funciones

Podemos llamar (invocar) funciones dentro de otras funciones.

example.js

```
// Function definition
function getAgeInDays(age) {
 return age * 365;
}

// Function definition
function createGreeting(name, age) {
 // Function invocation
 const ageInDays = getAgeInDays(age);
 const message = "My Name is " + name + " and I
was born over " + ageInDays + " days ago!";
 return message;
}

// Function invocation
console.log(createGreeting("Alexandra", 31));


// How is the execution thread?
```

JS Funciones

Podemos llamar (invocar) funciones dentro de otras funciones.

example.js

```
// Function definition
function getAgeInDays(age) {
 return age * 365;                                          3
}

// Function definition
function createGreeting(name, age) {
 // Function invocation
 const ageInDays = getAgeInDays(age);                       2
 const message = "My Name is " + name + " and I             4
was born over " + ageInDays + " days ago!";
 return message;                                            5
}

// Function invocation
console.log(createGreeting("Alexandra", 31));               1
```

JS Funciones

Podemos llamar (invocar) funciones dentro de otras funciones.

example.js

```
// Function definition
function getAgeInDays(age) {
 return age * 365;                                    3
}

// Function definition
function createGreeting(name, age) {
 // Function invocation
 const ageInDays = getAgeInDays(age);                 2
 const message = "My Name is " + name + " and I       4
was born over " + ageInDays + " days ago!";
 return message;                                      5
}

// Function invocation
console.log(createGreeting("Alexandra", 31));        1/6
```

Ejercicios

Ejercicio I (20 minutos)

1. Crear un archivo llamado `exercise-I.js` en el directorio `week-1/InClass`
2. Escribe una función que retorne el año en el que nació una persona dada su edad como parámetro
3. Usando la respuesta del paso 2, escribe una función que reciba como parámetro el nombre y la edad de una persona y retorne un string que contenga el nombre y el año de nacimiento de esa persona.

JS Alcances (Scope)

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
 let y = "local y";
 console.log(x); // x is reachable
}

foo();

console.log(y); // ?

{
 let y = "local y";
 console.log(x); // x is alcanzable
}

console.log(y); // ?
```

JS Alcances (Scope)

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
 let y = "local y";
 console.log(x); // x is reachable
}

foo();

console.log(y); // ERROR y is undefined

{
 let y = "local y";
 console.log(x); // x is alcanzable
}

console.log(y); // ERROR y is undefined
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
 let x = "local y";
 console.log(x); // ?
}

foo();

console.log(x); // ?

{
 let x = "local y";
 console.log(x); // ?
}

console.log(x); // ?
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
 let x = "local y";
 console.log(x); // local x
}

foo();

console.log(x); // global x

{
 let x = "local y";
 console.log(x); // local x
}

console.log(x); // global x
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
  let x = "local x";
  console.log(x); // ?

  {
    console.log(x); // ?
  }
}

foo();

console.log(x); // ?
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
  let x = "local x";
  console.log(x); // local x

  {
    console.log(x); // local x
  }
}

foo();

console.log(x); // global x
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    let x = "local x";
    console.log(x); // ?

    {
        let x = "nested local x";
        console.log(x); // ?
    }
    console.log(x); // ?
}

foo();

console.log(x); // ?
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    let x = "local x";
    console.log(x); // local x

    {
        let x = "nested local x";
        console.log(x); //nested local x
    }
    console.log(x); // local x
}

foo();

console.log(x); // global x
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
    let x = "local x";
    console.log(x); // local x

    {
        let x = "nested local x";
        console.log(x); //nested local x
    }
    console.log(x); // local x
}

foo();

console.log(x); // global x
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
      x = "local x";
      console.log(x); // ?

      {
            x = "nested local x";
            console.log(x); // ?
      }
      console.log(x); // ?
}

foo();

console.log(x); // ?
```

JS Alcances

Alcance:

Define en qué partes del código una variable está disponible (leer/escribir)

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
      x = "local x";
      console.log(x); // local x

      {
            x = "nested local x";
            console.log(x); //nested local x
      }
      console.log(x); // nested local x
}

foo();

console.log(x); // nested local x
```

JS Alcances

Alcance (usando var):

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
 var y = "local y";
 console.log(y); // local y
}

foo();

console.log(y); // ?

{
 var y = "local y";
 console.log(y); // local y
}

console.log(y); // ?
```

JS Alcances

Alcance (usando var):

- Global
- Function (local)
- Block (local)

example.js

```
let x = "global x";

function foo() {
 var y = "local y";
 console.log(y); // local y
}

foo();

console.log(y); // ERROR y is undefined

{
 var y = "local y";
 console.log(y); // local y
}

console.log(y); // local y
```

JS Alzamiento (Hoisting)

El motor de JavaScript mueve (hoists) la definición de todas las funciones y las variables declaradas con var al inicio del script.

example.js

```
// Function invocation
greeting("Hello, my name is ", "Alexandra");

// Function definition
function greeting(input_name, greeting)  {
 var greeting = greetingStart + name;
 console.log(greeting);
}

x = "initializing before declaring x?"

var x;
```