



DESARROLLO WEB EN ENTORNO CLIENTE

UD2 – SINTAXIS DEL LENGUAJE JAVASCRIPT

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE
APLICACIONES WEB

I.E.S. HERMENEGILDO LANZ – 2022/2023

PROFESORA: VANESA ESPÍN

vespin@ieshlanz.es



Resultados de Aprendizaje y Criterios de Evaluación

RA2. Escribe sentencias simples, aplicando la sintaxis del lenguaje y verificando su ejecución sobre navegadores Web

CRITERIOS de evaluación del RA2	%UD	%Curso
a) Se ha seleccionado un lenguaje de programación de clientes Web en función de sus posibilidades.	10%	5%
b) Se han utilizado los distintos tipos de variables y operadores disponibles en el lenguaje.	15%	
c) Se han identificado los ámbitos de utilización de las variables.	15%	
d) Se han reconocido y comprobado las peculiaridades del lenguaje respecto a las conversiones entre distintos tipos de datos.	10%	
e) Se han añadido comentarios al código.	10%	
f) Se han utilizado mecanismos de decisión en la creación de bloques de sentencias.	15%	
g) Se han utilizado bucles y se ha verificado su funcionamiento.	15%	
h) Se han utilizado herramientas y entornos para facilitar la programación, prueba y depuración del código	10%	

Objetivos Didácticos de la unidad 2

1. Comprender la sintaxis básica de JavaScript para poder realizar pequeños scripts funcionales
2. Entender las reglas básicas del lenguaje JavaScript
3. Conocer el uso de las sentencias básicas JavaScript

IMPORTANTE: no todo lo que aprenderemos está en las diapositivas, HAY QUE SEGUIR LOS ENLACES DE REFERENCIA INDICADOS Y HACER LOS EJERCICIOS



Índice

10 reglas básicas de Javascript

Variables y tipos de datos

Ámbitos de las variables

Operadores

Conversión de tipos

Números aleatorios

Bloques de Código: Condicionales y Bucles

Preparación de Visual Studio

10 Reglas Básicas

- *Regla 1.* Las instrucciones en JavaScript terminan en un punto y coma. Ejemplo:

```
var s = "hola";
```

- *Regla 2.* Uso de decimales en JavaScript. Los números en JavaScript que tengan decimales utilizarán el punto como separador de las unidades con la parte decimal. Ejemplos de números:

```
var x = 4;  
var pi = 3.14;
```

- *Regla 3.* Los literales se pueden escribir entre comillas dobles o simples. Ejemplo:

```
var s1 = "hola";  
var s2 = 'hola';
```

- *Regla 4.* Cuando sea necesario declarar una variable, se utilizará la palabra reservada *var*.
- *Regla 5.* El operador de asignación, al igual que en la mayoría de lenguajes, es el símbolo igual (=).
- *Regla 6.* Se pueden utilizar los siguientes operadores aritméticos: (+ - * /) . Ejemplo:

o *let* o *const*

```
var x = (5*4)/2+1;
```


10 Reglas Básicas. continuación

- *Regla 7.* En las expresiones, también se pueden utilizar variables. Ejemplo:

```
var t = 4;  
var x = (5*t)/2+1;  
var y;  
y = x * 2;
```

- *Regla 8.* Comentarios en JavaScript. Existen dos opciones para comentar el código:

- a) `//` cuando se desea comentar el resto de la línea a partir de estas dos barras invertidas.
- b) `/*` y `*/`. todo lo contenido entre ambas etiquetas quedará comentado.

- *Regla 9.* Los identificadores en JavaScript comienzan por una letra o la barra baja (`_`) o el símbolo del dólar (`$`).
- *Regla 10.* JavaScript es sensible a las mayúsculas y minúsculas (case-sensitive). Ejemplo:

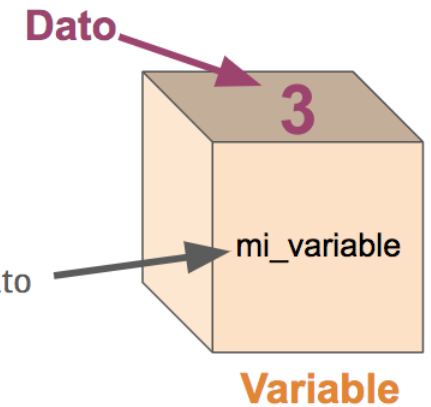
```
var nombre = "Julio";  
var Nombre = "Ramón";
```

Variables

posiciones de memoria para alojar datos

Variables JS

Cada variable tiene un nombre, de modo que podamos acceder a ese dato siempre que necesitemos.



- **Valores:** datos que el programa necesita almacenar, manipular o mostrar.
- Tipos de valores básicos: Textos (Strings), Números, Booleanos
- Operador ***typeof***: para conocer el tipo de un valor

```
typeof 2      5 'number'  
typeof true   6 'boolean'  
typeof 'hola' 7 'string'  
typeof undefined 8 'undefined'
```

Si no sabe de qué tipo es una variable, `typeof` devolverá *undefined*

https://www.w3schools.com/js/js_variables.asp

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_types#variables

Declaración de variables: var, let y const

necesario para poder usar nuestras variables

- Tradicionalmente mediante la palabra reservada *var*

```
var x=9;  
x=25;  
console.log(x);
```
- *let* y *const* aparecen en la versión 6 del estándar ECMAScript (o ES6).
- *let* : para declarar variables de forma más local que *var*. Restringe su uso al bloque donde se encuentra

```
{  
    let x=9;  
}  
console.log(x); //Devolverá error -> ReferenceError: x is not defined
```


var versus let

- *var* es de ámbito global y *let* es de ámbito local.
- ¿Cuál es el resultado de los siguientes códigos?

```
var x=7;  
{  
    var x=9;  
}  
console.log(x);
```

Devolverá x=9

```
let x=7;  
{  
    let x=9;  
}  
console.log(x);
```

Devolverá x=7

*Aunque en realidad no se debe hacer esto
Ya que let no permite redeclarar variables*

Declaración *const*

- Para definir constantes.
- Su ámbito es igual que el de *let*.

Constantes en JavaScript

En JavaScript, a partir de ES6 (ES2015), se puede utilizar la palabra reservada *const*. Por lo tanto, en vez de utilizar:

```
var pi = 3.141592;
```

Se aconseja emplear:

```
const PI = 3.141592;
```

Dado que *pi* es una constante (valor invariable). Por convención, se suele utilizar mayúsculas cuando se definen constantes.

- No podrá modificarse su valor durante el resto del programa.





```
const PI=3.1416;  
PI=9; //Al ejecutar el código dará error --> TypeError: Assignment to constant variable
```

Ámbito de las Variables (Scope)

- Cuando declaras una variable fuera de cualquier función, se denomina variable **global**, porque está disponible para cualquier otro código en el documento actual.
 - En las páginas web, el objeto global es **window**, por lo que puedes establecer y acceder a variables globales utilizando la sintaxis **window.variable**.
- Cuando declaras una variable dentro de una función, se llama variable **local**, porque solo está disponible dentro de esa función.
- Cuando declaras una variable dentro de un bloque y solo se conoce dentro de ese bloque, se dice que tiene ámbito de **bloque**.

ENLACE DE REFERENCIA: https://www.w3schools.com/js/js_scope.asp

VAR vs LET vs CONST

	var	let	const
Stored in Global Scope			
Function Scope			
Block Scope			
Can Be Reassigned?			
Can Be Redeclared?			

Hoisting o elevación

- Comportamiento predeterminado de JavaScript de mover **declaraciones** a la parte superior del alcance (scope) actual.
 - El uso de una variable con *let* antes de declararla dará como resultado un archivo *ReferenceError*.
 - Usar una variable con *const* antes de declararla es un error de sintaxis, por lo que el código simplemente no se ejecutará.

Esto dará como resultado un **ReferenceError** :

```
carName = "Volvo";  
let carName;
```

Este código no se ejecutará.

```
carName = "Volvo";  
const carName;
```

- Las **inicializaciones no se elevan** pero llevan a undefined

Para evitar errores, siempre declara todas las variables al comienzo de cada ámbito.

- ENLACE A SEGUIR: https://www.w3schools.com/js/js_hoisting.asp

Tipos de Datos

- El último estándar ECMAScript define 8 tipos de datos.
 - Siete tipos de datos que son primitivos:
 1. Booleano. `true` y `false`.
 2. null. Una palabra clave especial que denota un valor nulo. (Dado que JavaScript distingue entre mayúsculas y minúsculas, `null` no es lo mismo que `Null`, `NULL` o cualquier otra variante).
 3. undefined. Una propiedad de alto nivel cuyo valor no está definido.
 4. Number. Un número entero o un número con coma flotante. Por ejemplo: `42` o `3.14159`.
 5. BigInt. Un número entero con precisión arbitraria. Por ejemplo: `9007199254740992n`.
 6. String. Una secuencia de caracteres que representan un valor de texto. Por ejemplo: `"Hola"`.
 7. Symbol (nuevo en ECMAScript 2015). Un tipo de dato cuyas instancias son únicas e inmutables
 - y Object `var persona = {nombre:"Dimas", apellido:"Moreno"}; // Objeto`

nuevos

Literales

Se utilizan para representar valores en JavaScript. Son valores fijos, no variables, que como su nombre indica son literalmente proporcionados por el programador en el código.

NÚMEROS

- ya que no diferencia entre decimales y enteros todos ocupan el mismo espacio en memoria (64 bits).

```
let entero=1980;  
let decimal=0.21;
```

- Formatos de números:
 - Notación científica
 - Hexadecimal
 - Octal
 - Binario
 - Especiales: Infinity y NaN

```
let AVOG=6.022e+23      //6,022 · 10^23  
let hexa=0xAB12;        //43794  
let octal=0o27652;      //12202  
let binario=0b10111011; //187  
  
var x=1/0;               //Infinity  
var x="Hola" * 3         //NaN
```

Literales

STRINGS

- Se permite delimitar cadenas de caracteres entre comillas simples o dobles, así dentro del texto delimitado pueden aparecer comillas dobles o simples

```
frase1="Mi apellido es O'Donnell";  
frase2='José se acercó y me dijo "Hola"'
```

- Puedo concatenar cadenas incluso variables usando el operador +

```
var nombre="Juan";  
console.log("Me llamo "+nombre);
```

- En la versión 6 aparecen las comillas invertidas ` que permiten aplicar las **Plantillas de String** (*String Template*) usando el símbolo `\${ }` e incluyendo entre las llaves la variable a evaluar.

```
var nombre="Pepe";  
console.log(`Me llamo ${nombre}`);
```

Literales

STRINGS – Secuencias de escape

Útil para caracteres que no se pueden escribir directamente en un String

Escape	Efecto
"\\texto de prueba"	texto de prueba
"texto \"de\" prueba"	texto "de" prueba
"texto \'de\' prueba"	texto 'de' prueba

Secuencia de escape	Descripción
\n	Nueva línea. Se establece el curso en la próxima línea de la pantalla
\t	Tabulación. Mueve el cursor a la próxima posición de tabulación.
\r	Retorno. Mueve el cursor hacia el inicio de la línea actual
\a	Alerta. Se produce un sonido de alerta en el sistema
\\	Barra diagonal inversa. Se imprime la barra diagonal inversa
\'	Comilla sencilla. Se usa para imprimir una comilla sencilla
\"	Comilla doble. Se usa para imprimir una comilla doble

Literales

BOOLEANOS

- Solo pueden tomar los valores *true* y *false*

```
let verdad=true;
```

- Es habitual que se produzcan al evaluar expresiones

```
let x=9;  
let y=10;  
let mayor=(x>y); //mayor valdrá false
```

- La función Boolean nos devuelve el valor booleano equivalente a cualquier valor. Ejemplos

```
console.log (Boolean(1)); //Escribe true  
console.log (Boolean(0)); //Escribe false  
console.log (Boolean("hola")); //Escribe true  
console.log (Boolean("")); //Escribe false
```

```
console.log (Boolean(undefined)); //Escribe false  
console.log (Boolean(null)); //Escribe false  
console.log (Boolean(Infinity)); //Escribe true
```

Operadores

https://www.w3schools.com/js/js_operators.asp

Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++	Incremento
--	Decremento

Operadores de asignación

Operador	Ejemplo de uso
=	x = y;
+=	x += y; // igual que (x = x + y)
-=	x -= y; // igual que (x = x - y)
*=	x *= y; // igual que (x = x * y)
/=	x /= y; // igual que (x = x / y)
%=	x %= y; // igual que (x = x % y)

Operadores de manejo de strings

En JavaScript, se utilizan los operadores `+` y `+=` para concatenar strings. Véase un ejemplo de uso:

```
var = "hola" + " " + "mundo";
```

O lo que sería igual:

```
var = "hola";  
var += " ";  
var += "mundo";
```

Operadores de tipo

Los operadores de tipo permiten conocer si un objeto es una instancia de un tipo concreto o bien conocer el tipo de una variable. A continuación, se muestran los operadores de tipo disponibles en JavaScript:

- *typeof*. Devuelve el tipo de una variable.
- *instanceof*. Devuelve true si un objeto es una instancia de un tipo de objeto.

Operadores lógicos y de comparación

Operador	Descripción
==	Igual que
===	Igual valor y tipo
!=	Distinto
!==	Distinto valor o distinto tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
?	Operador ternario

Comparación estricta: === y !==

```
console.log ("2"==2);    //Escribe true
console.log ("2"===2);   //Escribe false
console.log (1+1===2);   //Escribe true
console.log ("2"!=2);    //Escribe false
console.log ("2"!==2);   //Escribe true
```

El valor null

Null para los lenguajes de programación es *nada* o algo que no existe. Generalmente, cuando se asigna null a una variable, es porque es o será un objeto.

Véase un ejemplo de uso del null:

```
var persona = null;  
persona = {nombre:"Dimas", apellido:"Moreno"};
```

Cuando evalúas una variable `null`, el valor nulo se comporta como `0` en contextos numéricos y como `false` en contextos booleanos. Por ejemplo:

```
var n = null;  
console.log(n * 32); // Registrará 0 en la consola
```

El valor undefined

Puedes usar `undefined` para determinar si una variable tiene un valor. En el siguiente código, a la variable `input` no se le asigna un valor y la declaración `if` evalúa a `true`.

```
var input;  
if (input === undefined) {  
    doThis();  
} else {  
    doThat();  
}
```

El valor `undefined` se convierte en `NaN` cuando se usa en contexto numérico.

```
var a;  
a + 2; // Evalúa a NaN
```

Not-A-Number

Conversiones entre tipos de datos

- JavaScript es de tipado dinámico. Conversiones de tipo son transparentes.

```
var answer = 42;  
answer = 'Gracias por todo el pescado...';
```

- El operador + permite convertir los operadores numéricos en cadenas cuando hay alguna cadena implicada

```
x = 'La respuesta es ' + 42 // "La respuesta es 42"  
y = 42 + ' es la respuesta' // "42 es la respuesta"
```

OJO! →

```
'37' - 7 // 30  
'37' + 7 // "377"
```

- También existen funciones de conversión desde texto a entero o flotante:

- `parseInt()`
- `parseFloat()`

```
parseInt('101', 2) // 5
```

↑
radix (base?)

Conversiones entre tipos de datos

- El método toString() convierte números en cadena de caracteres:

```
let num = 999;  
num.toString();//Devolverá "999" como cadena de  
caracteres  
(888+111).toString() //Devolverá?...
```

- Seguir por ENLACE:
https://www.w3schools.com/js/js_type_conversion.asp

Números aleatorios

- Requieren el uso del objeto **Math** (que se explicará en otro tema).
- Generación de número aleatorio. Ejemplos:

```
Math.random();           //Decimal entre 0 (incluido) y 1 (excluido)
Math.random()*2;         //Decimal entre 0 y 2
Math.random()*4+6;       //Decimal entre 6 y 10

parseInt(Math.random()*11); //Entero entre 0 y 10 (ambos incluidos)
Math.floor(Math.random()*11); //Lo mismo
```

- ¿Entero entre 6 y 10?

```
Math.floor(Math.random()*5)+6; //Entero entre 6 y 10 (ambos incluidos)
```


Sentencias condicionales. Sintaxis

https://www.w3schools.com/js/js_if_else.asp

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

```
if (condition) {  
    // block of code to be executed if the condition is true  
} else {  
    // block of code to be executed if the condition is false  
}
```


```
if (condition1) {  
    // block of code to be executed if condition1 is true  
} else if (condition2) {  
    // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    // block of code to be executed if the condition1 is false and condition2 is false  
}
```

https://www.w3schools.com/js/js_switch.asp

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

Bucles. Sintaxis (hay más tipos de bucles que veremos más adelante)


```
for (expression 1; expression 2; expression 3) {  
    // code block to be executed  
}
```



```
for (let i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```


https://www.w3schools.com/js/js_loop_for.asp

```
while (condition) {  
    // code block to be executed  
}
```



```
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

```
do {  
    // code block to be executed  
}  
while (condition);
```



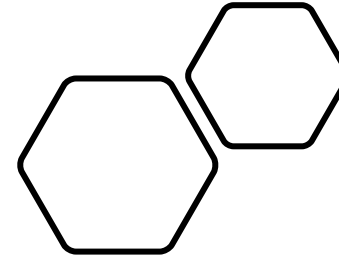
```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```

https://www.w3schools.com/js/js_loop_while.asp

Break para salir de un bucle
Continue para saltar una iteración
https://www.w3schools.com/js/js_break.asp

Práctica 1

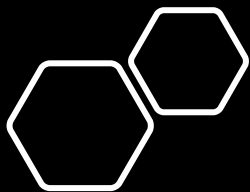
Configurando Visual Studio Code



1. Añadiremos extensiones
2. Instalaremos node.js
3. Instalaremos nvm (control de versiones para node.js)
4. Instalaremos Git y lo usaremos



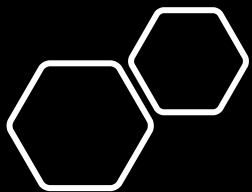
Visual Studio Code



Instalación de extensiones

En el panel de extensiones,
apartado **Habilitado**, vemos las
extensiones instaladas

- Desde el panel de extensiones instalaremos las siguientes:
 1. Auto Rename Tag: para que al modificar etiqueta se modifique también la de cierre
 2. Open in Browser: para ejecutar en navegador
 3. Live Server: muy potente. Nos crea un servidor local con ruta <http://127.0.0.1:5500>



Instalación de node.js

- En Windows: Ir a <https://nodejs.org/es/> y descarga la versión LTS. Instálalo. (Con las opciones por defecto).
- En Linux: seguir enlace <https://nodejs.org/en/download/package-manager/> y buscar nuestro sistema
- Probar node.js en terminal

```
C:\Users\Vanesa>node
Welcome to Node.js v16.17.1.
Type ".help" for more information.
> console.log("Hola");
Hola
undefined
>
```

Salimos con CTRL+C y escribimos:

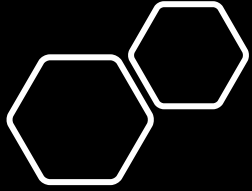
```
C:\Users\Vanesa>code prueba1.js
```

JS prueba1.js X

C: > Users > Vanesa > JS prueba1.js

```
1 console.log ("HOLA desde este programa");
```

```
C:\Users\Vanesa>node prueba1.js
HOLA desde este programa
```



Instalación de nvm

El administrador de versiones de Node, más comúnmente denominado nvm

- **Desinstalar primero node.js** de modo convencional.
- En Windows: Ir a <https://github.com/coreybutler/nvm-windows/releases> y descargar *nvm-setup.exe*. Instálalo. (Existe otro sitio en github para nvm en Linux).

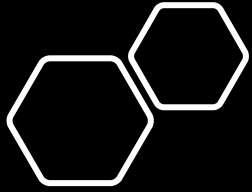
```
C:\Users\Vanesa>nvm ls available
```

```
C:\Users\Vanesa>nvm install latest
```

```
C:\Users\Vanesa>nvm install latest  
Version 18.9.1 is already installed.
```

```
C:\Users\Vanesa>nvm use 18.9.1
```

- Volver a probar node.js desde línea de comandos
- Instala ahora dos versiones antiguas (la 4 y la 10) y comprueba las versiones instaladas. (*nvm install* y *nvm ls*)
- Fuerza a usar la versión 4 (*nvm use*) y compruebalo (*node -v*)
- Vuelve a usar la versión 18 y desinstala la versión 4
- Comprueba al resultado final (versiones instaladas y versiones en uso)



Instalación de Git

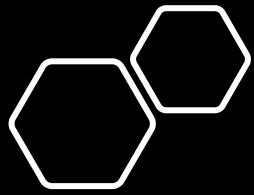
1. Vamos a <https://git-scm.com/> descargamos nuestra versión y la instalamos con las siguientes opciones:
 - Elegimos Visual Studio Code como editor por defecto de Git
 - PATH: elegimos la tercera opción (optional Linux Tools)
 - Librería OpenSSL
 - Terminal MinTTY
2. Comprobamos la versión instalada



```
C:\Users\Vanesa>git --version  
git version 2.37.3.windows.1
```

3. Configuramos algunos parámetros globales:

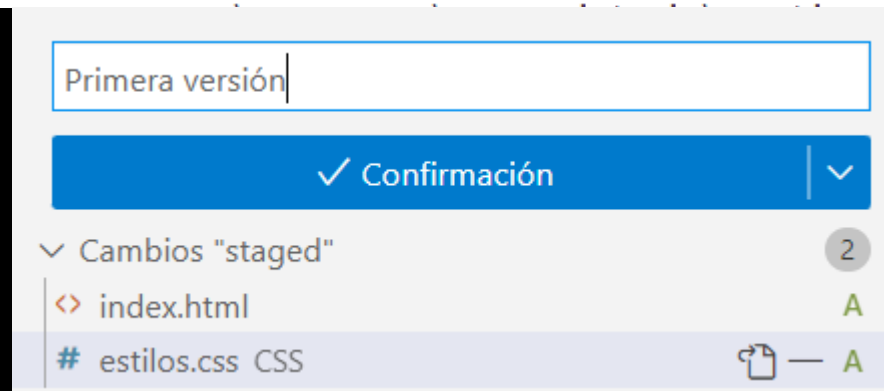
```
C:\Users\Vanesa>git config --global user.name "Vanesa Espin"  
C:\Users\Vanesa>git config --global user.email "vespin@ieshlanz.es"
```



Uso de Git para crear versiones

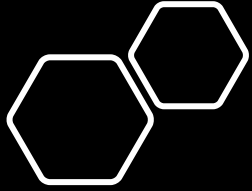
1. Crear una carpeta llamada Practica1 en tu carpeta de este tema
2. Crea dentro un archivo *index.html* de título “Practica 1 sobre Git” que incluya de body un título `<h1>` con el mismo texto
3. Crea el archivo *estilos.css* dentro de una carpeta CSS donde le demos estilos a h1 (tipo de letra y color).
4. CTRL+ñ para ir al terminal de VSCode y en la raíz de nuestra carpeta de trabajo escribimos: *git init*

```
PS V:\0-DOCENCIA\0-2022-IES-HLANZ\2-DWEC BIL\UD2 - Sintaxis\Practica1> git init
Initialized empty Git repository in V:/0-DOCENCIA/0-2022-IES-HLANZ/2-DWEC BIL/UD2 - Sintaxis/Practica1/.git/
```



confirmamos

5. Si quieres Instala la extensión GitLens y haz pruebas de commits y checkouts



Subir nuestro código a GitHub

1. Crear una cuenta en GitHub (si no la tienes)
2. Crea un nuevo repositorio con el nombre que quieras, por ejemplo ***practica1***.
3. Sube tu código al nuevo repositorio (recuerda que puedes usar comandos desde el terminal de VS Code *CTRL+ñ*).
4. Crea cambios en tu código y publica alguna rama.

Algunos comandos útiles

```
git remote add origin https://github.com/tunombre/turepositorio.git  
git branch -M main  
git push -u origin main
```

ENSEÑA TU PRÁCTICA A LA PROFESORA CUANDO TE FUNCIONE



The End