

# JAVASCRIPT

UF1842

---



# ¿Qué es JavaScript?



JavaScript fue creado para “dar vida a las páginas web”.



Los programas en este lenguaje se llaman *scripts*. Se pueden escribir directamente en el HTML de una página web y ejecutarse automáticamente a medida que se carga la página.



Los scripts se proporcionan y ejecutan como texto plano. No necesitan preparación especial o compilación para correr.

# El motor o intérprete de JavaScript

---


V8 – en Chrome, Opera y Edge.

---

SpiderMonkey – en Firefox.

---

...Existen otros nombres en clave como “Chakra” para IE ,  
“JavaScriptCore”, “Nitro” y “SquirrelFish” para Safari, etc.

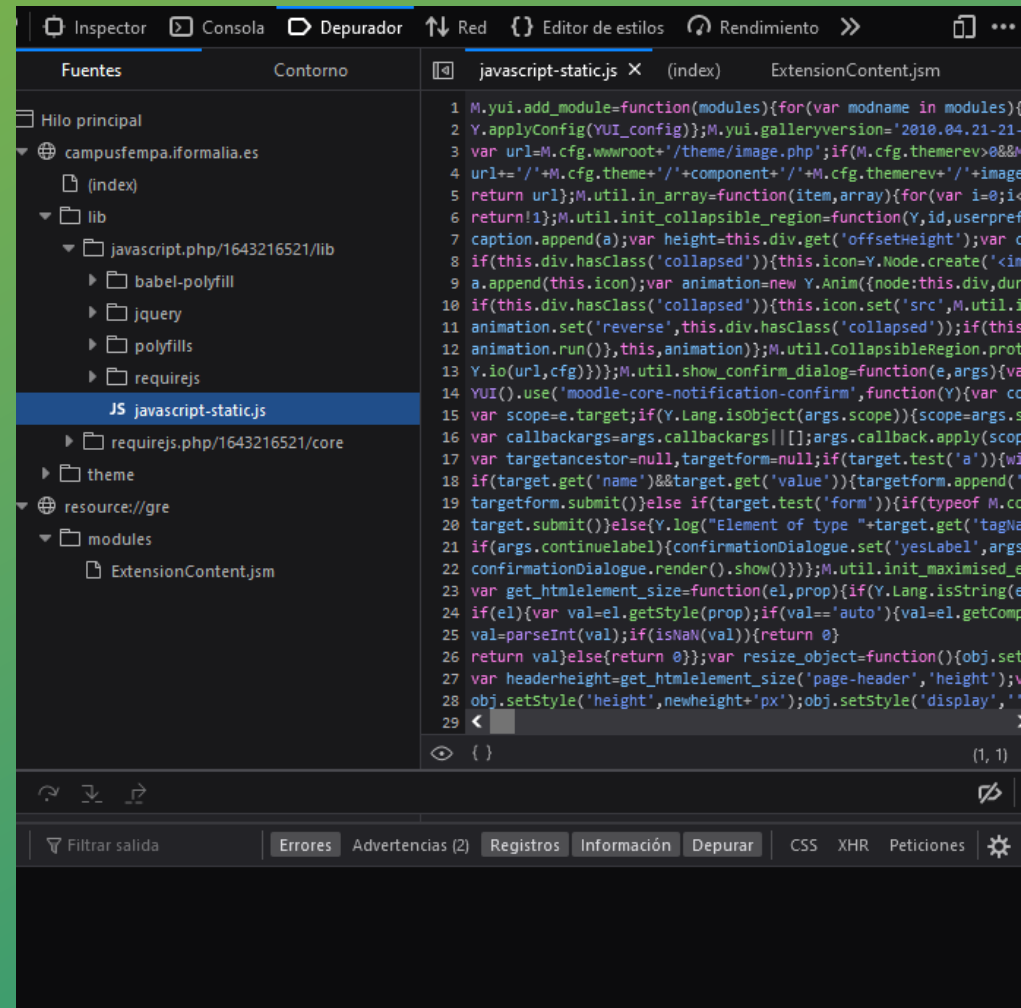


# Especificaciones y sitios de ayuda

- La especificación ECMA-262 contiene la información más exhaustiva, detallada, y formal sobre JavaScript. En ella se define el lenguaje.
- **MDN (Mozilla) JavaScript Reference** es el manual principal, con ejemplos y otras informaciones. Es fantástico para obtener información exhaustiva sobre funciones individuales del lenguaje, métodos, etc. Se puede acceder en <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference>.

# CONSOLA DE DESARROLLADOR

Presione F12 o, si está en Mac,  
entonces combine Cmd+Opt+J.



# Entrada multilínea

Por lo general, cuando colocamos una línea de código en la consola y luego presionamos Enter, se ejecuta.

Para insertar varias líneas, presione Shift+Enter. De esta forma se pueden ingresar fragmentos largos de código JavaScript.

# Ejecutar JavaScript



Para entornos del lado del servidor (como Node.js), puedes ejecutar el script con un comando como:

- "node miScript.js"

Para el navegador:

- Visualizar la página html

# Uso de JavaScript



## La etiqueta <script>

- Los programas de JavaScript se pueden insertar en casi cualquier parte de un documento HTML con el uso de la etiqueta <script>

## Scripts externos:

- Los archivos de script se adjuntan a HTML añadiendo el atributo `src="..."` a la etiqueta.



# La etiqueta “script”

- El atributo type="text/javascript". Ya no es necesario.
- El atributo language estaba destinado a mostrar el lenguaje del script. Este atributo ya no tiene sentido.
- Una sola etiqueta <script> no puede tener el atributo src y código dentro.

```
1  <!DOCTYPE HTML>
2  <html>
3
4  <body>
5
6    <p>Antes del script...</p>
7
8    <script>
9      alert( '¡Hola, mundo!' );
10   </script>
11
12   <p>...Después del script.</p>
13
14 </body>
15
16 </html>
```

+

•

○

# Ejercicios

1. Mostrar una alerta
  - Crea una página que muestre el mensaje “¡Soy JavaScript!”.
  - Hazlo en la consola o en tu disco duro, no importa, solo asegúrate de que funcione.
2. Mostrar una alerta con un script externo
  - Partiendo de la solución anterior. Modifícala extrayendo el contenido del script a un archivo externo alerta.js, ubicado en la misma carpeta.
  - Abre la página, para asegurarte que la alerta funciona.

# Estructura del código

## Sentencias

- Las sentencias son construcciones sintácticas y comandos que realizan acciones.

## Punto y coma

- Se puede omitir un punto y coma en la mayoría de los casos cuando existe un salto de línea.
- Colocar siempre puntos y coma entre las sentencias.

## Comentarios

- Los comentarios se pueden poner en cualquier lugar de un script. No afectan su ejecución porque el motor simplemente los ignora.
- Los comentarios de una línea comienzan con dos caracteres de barra diagonal //.
- Los comentarios de varias líneas comienzan con una barra inclinada y un asterisco /\* y terminan con un asterisco y una barra inclinada \*/.
- No puede haber /\*...\*/ dentro de otro /\*...\*/.

# El modo moderno, "use strict"



- ECMAScript 5 (ES5) apareció en 2009. Esta versión añadió nuevas características al lenguaje y modificó algunas de las ya existentes. Para mantener el código antiguo funcionando, la mayor parte de las modificaciones están desactivadas por defecto. Tienes que activarlas explícitamente usando una directiva especial: "use strict".

# “use strict”



Cuando se sitúa al principio de un script, el script entero funciona de la manera “moderna”.



No necesitamos agregar la directiva "use strict" si usamos cosas modernas como “clases” y “módulos”, que automáticamente habilitan use strict.

```
1  "use strict";  
2  
3  // este código funciona de la manera moderna  
4  ...
```



# Variables

- Podemos declarar variables para almacenar datos al utilizar las palabras clave **var**, **let**, o **const**.
  - **let** – es la forma moderna de declaración de una variable.
  - **var** – es la declaración de variable de vieja escuela. Normalmente no lo utilizamos en absoluto.
  - **const** – es como let, pero el valor de la variable no puede ser alterado.

# Variables



Introducimos datos en la variable al utilizar el operador de asignación =

Debemos declarar una variable una sola vez y desde entonces referirnos a ella sin let

Normas para nombrarlas:

- El nombre únicamente puede incluir letras, dígitos, o los símbolos \$ y \_
- El primer carácter no puede ser un dígito
- Cuando el nombre contiene varias palabras, comúnmente se utiliza camelCase
- Las mayúsculas son importantes
- Se puede usar la letra ñ, pero tradicionalmente sólo se usan las letras inglesas
- Hay una lista de palabras reservadas, las cuales no pueden ser utilizadas como nombre de variable porque el lenguaje en sí las utiliza. Por ejemplo: let, class, return, y function están reservadas
- Si no utilizamos "use strict", es posible crear una variable simplemente asignando un valor sin utilizar 'let', pero no hacerlo.



# Constantes

- Las variables declaradas utilizando `const` se llaman “constantes”. No pueden ser alteradas.
- Suelen nombrarse utilizando letras mayúsculas y guiones bajos. Ejemplo:

```
const COLOR_RED = "#F00";
```



# Ejercicios

## Trabajando con variables

- Declara dos variables: admin y nombre.
- Asigna el valor "Juan" a nombre.
- Copia el valor de nombre a admin.
- Muestra el valor de admin usando alert (debe salir "Juan").

## Dando el nombre correcto

- Crea una variable con el nombre de nuestro planeta. ¿Cómo nombrarías a dicha variable?
- Crea una variable para almacenar el nombre del usuario actual de un sitio web. ¿Cómo nombrarías a dicha variable?

## ¿const mayúsculas?

- Examina el siguiente código:  

```
const cumple = '18.04.1982';  
const edad = cualquierCodigo(cumple);
```
- Aquí tenemos una constante cumple y edad es calculada desde cumple con la ayuda de cierto código.
- ¿Sería correcto usar mayúsculas para cumple? ¿Para edad? ¿O incluso para ambos?  

```
const CUMPLE = '18.04.1982'; // ¿en mayúsculas?  
const EDAD = cualquierCodigo(CUMPLE); // ¿en mayúsculas?
```

# Tipos de datos

Un valor en JavaScript siempre pertenece a un tipo de dato determinado. Hay ocho tipos de datos básicos en JavaScript.

---

**number** para números de cualquier tipo: enteros o de punto flotante, los enteros están limitados por  $\pm(2^{53}-1)$ .

---

**bigint** para números enteros de longitud arbitraria.

---

**string** para cadenas. Una cadena puede tener cero o más caracteres, no hay un tipo especial para un único carácter.

---

**boolean** para verdadero y falso: true/false.

---

**null** para valores desconocidos – un tipo independiente que tiene un solo valor nulo: null.

---

**undefined** para valores no asignados – un tipo independiente que tiene un único valor “indefinido”: undefined.

---

**object** para estructuras de datos complejas.

---

**symbol** para identificadores únicos.

---

# El operador typeof

Dos formas: `typeof x` o `typeof(x)`.

Devuelve una cadena con el nombre del tipo. Por ejemplo "string".

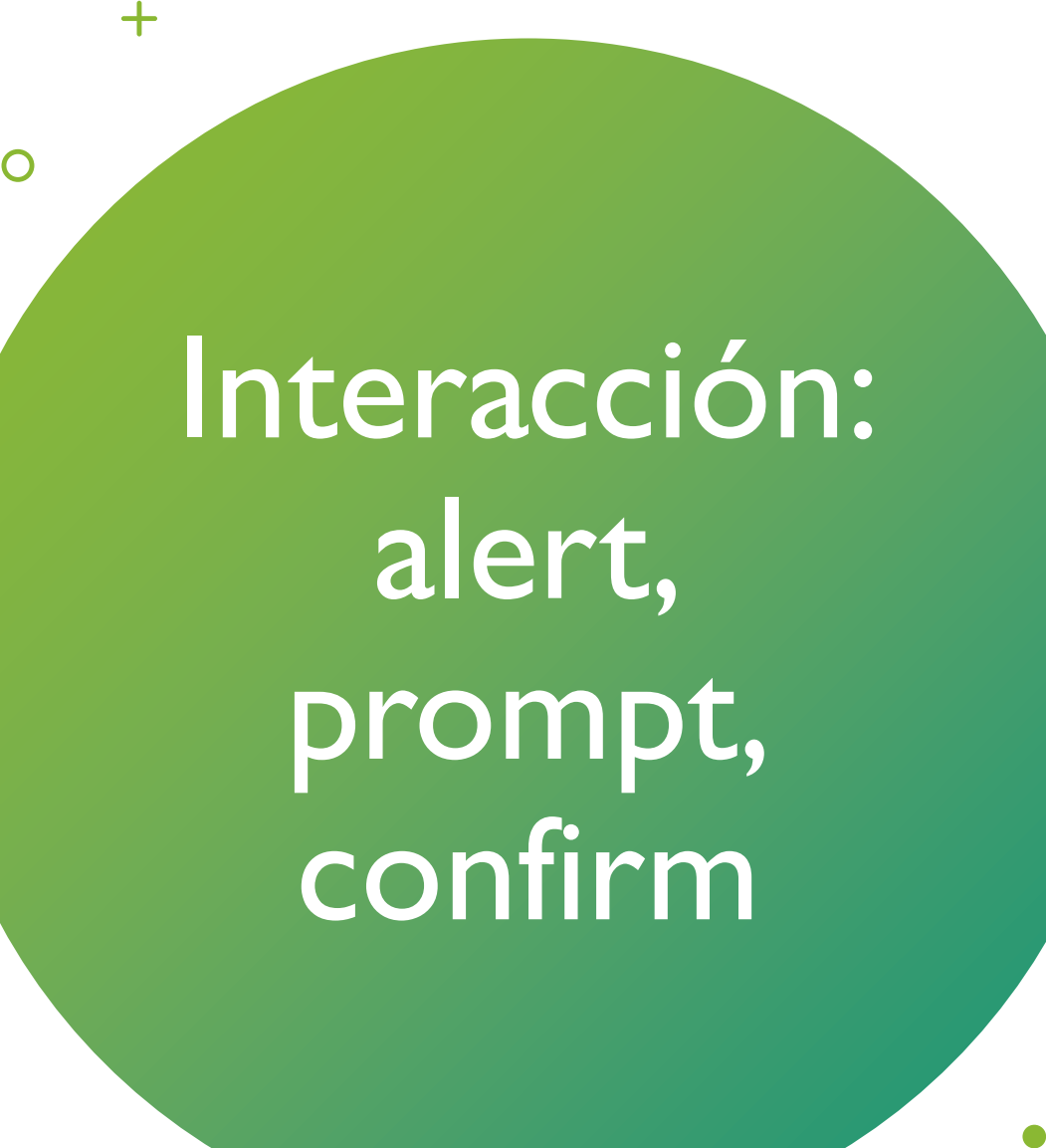
Para null devuelve "object": esto es un error en el lenguaje, en realidad no es un objeto.

# Ejercicios

## 1. Comillas - *backticks* (`)

- ¿Cuál es la salida del script?

```
1  let name = "Ilya";
2
3  alert( `Hola ${1}` ); // ?
4
5  alert( `Hola ${"name"}` ); // ?
6
7  alert( `Hola ${name}` ); // ?
```



# Interacción: alert, prompt, confirm

- Como usaremos el navegador como nuestro entorno de demostración, tenemos unas funciones para interactuar con el usuario.
- Todos estos métodos son modales: detienen la ejecución del script y no permiten que el usuario interactúe con el resto de la página hasta que la ventana se haya cerrado.
- Hay dos limitaciones:
  - La ubicación exacta de la ventana.
  - El aspecto de la ventana.

# Interacción: alert, prompt, confirm

alert

- muestra un mensaje.

prompt

- muestra un mensaje pidiendo al usuario que introduzca un texto. Retorna el texto o, si se hace clic en CANCELAR o Esc, retorna null.

confirm

- muestra un mensaje y espera a que el usuario pulse “OK” o “CANCELAR”. Retorna true si se presiona OK y false si se presiona CANCEL/Esc.

+

•

○

# Ejercicios

1. Una página simple
  - Crea una página web que pida un nombre y lo muestre usando las funciones que acabamos de ver.

# Conversiones de Tipos

La mayoría de las veces, los operadores y funciones convierten automáticamente los valores que se les pasan al tipo correcto. Esto es llamado “conversión de tipo”. Por ejemplo, `alert` convierte automáticamente cualquier valor a string para mostrarlo. Las operaciones matemáticas convierten los valores a números.

La conversión explícita es requerida usualmente cuando leemos un valor desde una fuente basada en texto, como lo son los campos de texto en los formularios, pero que esperamos que contengan un valor numérico.



# Conversiones más comunes



Al tipo String – Ocurre cuando se muestra algo. Se puede realizar con **String(valor)**. La conversión a *string* es usualmente obvia para los valores primitivos.



Al tipo Number – Ocurre en operaciones matemáticas. Se puede realizar con **Number(valor)**.



Al tipo Boolean – Ocurren en operaciones lógicas. Se puede realizar con **Boolean(valor)**.

# REGLAS PARA NUMBER

| Valor        | Se convierte en...   |
|--------------|--|
| undefined    | NaN  |
| null         | 0  |
| true / false | 1 / 0  |
| string       | El string es leído "como es", los espacios en blanco tanto al inicio como al final son ignorados. Un string vacío se convierte en 0 . Un error entrega NaN . |

**Valor**

**Se convierte en.**

0, null, undefined, NaN, ""

false

cualquier otro valor

true

+

•

○

# REGLAS PARA BOOLEAN

# OPERADORES BÁSICOS, MATEMÁTICOS

Suma +

Resta -

+  
o

Multiplicación \*

División /

Resto % (es el  
resto de la división  
entera de a por b)

Exponenciación \*\*

# Operadores unarios, binarios...

- `let x = 1, y = 2, z = 3;`

```
/* ejemplo operador unario */
```

```
x = -x;
```

```
console.log( x ); // -1, se aplicó negación unaria
```

```
/* ejemplo operador binario */
```

```
console.log( z - y ); // 1, resta de valores
```

# Atención



Casi todas las operaciones matemáticas convierten valores a números. La suma `+` es especial, si uno de los valores sumados es un *string*, el otro valor es convertido a *string*. Luego, los concatena (une).



```
alert( 1 + '2' ); // '12' (string a la derecha)  
alert( '1' + 2 ); // '12' (string a la izquierda)
```

# Atención



El operador unario + se puede usar como alternativa a Number(), ya que produce el mismo resultado y es más corto de escribir

```
// Sin efecto en números
let x = 1;
alert(+x); // 1
let y = -2;
alert(+y); // -2
// Convierte los no números
alert(+true); // 1
alert(+""); // 0
```

# Más sobre operadores



## Precedencia

- Determina el orden en el cual los operadores son evaluados. Los operadores con mayor precedencia son evaluados primero.

## Asociatividad

- Determina el orden en el cual los operadores con el mismo nivel de precedencia son procesados.



# Ejercicios

## Tabla de Precedencia

- A estas alturas ya debemos saber dónde encontrar esta tabla.
- Échale un vistazo rápido.

## Asociatividad

- ¿Qué tipo de asociatividad tienen los operadores de multiplicación, división y módulo?
- ¿Qué tipo de asociatividad tienen los operadores de asignación?

## ¿Sabes el valor final de las cuatro variables?

- Intenta averiguarlo por ti mismo y comprueba tu solución con ayuda de la consola  

```
let a=1, b=2, c=3, d=4;  
a *= b = c = d--;
```

# Ejercicios

## Sufijo y prefijo

- ¿Cuáles son los valores finales de todas las variables a, b, c y d?

```
let a = 1, b = 1;  
let c = ++a; // ?  
let d = b++; // ?
```

## Asignación

- ¿Cuáles son los valores de 'a' y 'x'?

```
let a = 2;  
let x = 1 + (a *= 2);
```

# Ejercicios

## Conversiones de tipos

- ¿Cuáles son los resultados de estas expresiones?

- 1) "" + 1 + 0
- 2) "" - 1 + 0
- 3) true + false
- 4) 6 / "3"
- 5) "2" \* "3"
- 6) 4 + 5 + "px"
- 7) "\$" + 4 + 5
- 8) "4" - 2
- 9) "4px" - 2
- 10) " -9 " + 5
- 11) " -9 " - 5
- 12) null + 1
- 13) undefined + 1
- 14) " \t \n" - 2

## Corregir la suma

- Aquí hay un código que le pide al usuario dos números y muestra su suma. Funciona incorrectamente. El resultado en el ejemplo es 12. ¿Por qué? Arréglalo.

```
let a = prompt("¿Primer número?", 1);
let b = prompt("¿Segundo número?", 2);

alert(a + b); // 12
```

# Comparaciones



Los operadores de comparación retornan un valor booleano.

El doble signo `==` es la comparación.

El distinto es `!=` dado que el símbolo `≠` no existe.

Las cadenas se comparan letra por letra en orden “lexicográfico”.

Cuando se comparan valores de diferentes tipos, se convierten en números (excepto para el operador de igualdad estricta `===`).

Los valores **null** y **undefined** son iguales `==` entre sí y no equivalen a ningún otro valor.

Trata cualquier comparación con `undefined/null` (excepto la igualdad estricta `===`) con precaución.

No uses comparaciones `>=`, `>`, `<` o `<=` con una variable que pueden ser `null` o `undefined`, a menos que estés realmente seguro de lo que estás haciendo. Si una variable puede tener estos valores, verifícalos por separado.

# Ejercicios

## Comparaciones

- ¿Cuál será el resultado de las siguientes expresiones?

1) 5 > 4

2) "cola" > "cola-caa"

3) "2" > "12"

4) undefined == null

5) undefined === null

6) null == "\n0\n"

7) null === +"\n0\n"

# Condicionales



```
if (condición) sentencia1 [else sentencia2]
```

- Ejecuta una sentencia si una condición especificada es evaluada como verdadera (reglas de conversión). Si la condición es evaluada como falsa, otra sentencia puede ser ejecutada.

```
condición ? expr1 : expr2
```

- El operador condicional (ternario) es el único operador en JavaScript que tiene tres operandos. Este operador se usa con frecuencia como atajo para la instrucción if.

# Ejercicios

## if (un string con cero)

- ¿Se mostrará el alert?

```
if ("0") {  
  alert( 'Hola' );  
}
```

## Reescribe el 'if' como '?'

- Reescribe esta condición if usando el operador ternario '?':

```
let result;  
if (a + b < 4) {  
  result = 'Debajo';  
} else {  
  result = 'Encima';  
}
```

## Reescriba el 'if...else' con '?'

- Reescriba el if..else utilizando operadores ternarios múltiples?'. La legibilidad es importante, utiliza saltos de línea.

```
let message;  
if (login == 'Empleado') {  
  message = 'Hola';  
} else if (login == 'Director') {  
  message = 'Felicidades';  
} else if (login == '') {  
  message = 'Sin sesión';  
} else {  
  message = '';  
}
```

# Operadores lógicos



## || (OR)

- Si un operando no es un booleano, se lo convierte a booleano para la evaluación. Encuentra el primer valor verdadero.

## && (AND)

- Retorna true si ambos operandos son valores verdaderos y false en cualquier otro caso. Encuentra el primer valor falso.

## ! (NOT)

- Convierte el operando al tipo booleano: true/false. Retorna el valor contrario.



# Ejercicios

¿Cuál es el resultado?

- 1) `alert(null || 2 || undefined);`
- 2) `alert(alert(1) || 2 || alert(3));`
- 3) `alert(1 && null && 2);`
- 4) `alert(alert(1) && alert(2));`
- 5) `alert(null || 2 && 3 || 4);`

¿Cuáles de estos *alert* va a ejecutarse?

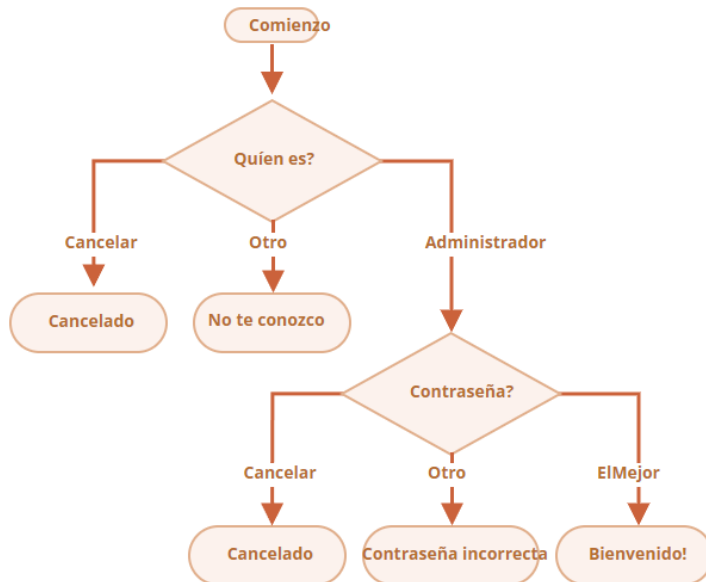
- 1) `if (-1 || 0) alert("primero");`
- 2) `if (-1 && 0) alert("segundo");`
- 3) `if (null || -1 && 1) alert("tercero");`

# Ejercicios

## Comprueba el inicio de sesión

- Escribe un código que pregunte por el inicio de sesión con prompt.
- Si el visitante ingresa "Admin", entonces prompt(pregunta) por una contraseña, si la entrada es una línea vacía o Esc – muestra "Cancelado.", si es otra cadena de texto – entonces muestra "No te conozco".
- La contraseña se comprueba de la siguiente manera:
  - Si es igual a "TheMaster", entonces muestra "Bienvenido!",
  - Si es otra cadena de texto – muestra "Contraseña incorrecta",
  - Para una cadena de texto vacía o una entrada cancelada, muestra "Cancelado."

## El esquema:



+

•

○

# Operador coalescente nulo ( ?? )

- Es un operador lógico que devuelve su operando del lado derecho cuando su operando del lado izquierdo es null o undefined, y de lo contrario devuelve su operando del lado izquierdo.
- Cuando se quiere asignar un valor predeterminado a una variable, un patrón común es usar el operador lógico OR ( || ):

```
let algo = '';  
let textoSimple = algo || 'Hola';  
// 'Hola'
```

- Sin embargo, debido a que || es un operador lógico booleano, el operando del lado izquierdo se fuerza a un booleano para la evaluación y no se devolverá ningún valor falso ( 0, "", NaN, null, undefined). Este comportamiento puede tener consecuencias inesperadas si considera **0**, "" o **NaN** como valores válidos.

```
let algo = '';  
let textoSimple = algo ?? 'Hola';  
// ''
```



# Bucles

- while: El while evaluará y transformará la condición a un booleano.

```
let i = 3;  
// cuando i sea 0, la condición  
se  
// volverá falsa y el bucle se  
detendrá  
while (i) {  
    alert(i);  
    i--;  
}
```



# Bucles

- do...while: El cuerpo siempre se ejecuta al menos una vez.

```
let i = 0;  
do {  
    alert(i);  
    i++;  
} while (i < 3);
```



# Bucles

- for: El más usado, consta de comienzo, condición, paso y cuerpo.

```
for (let i = 0; i < 3; i++) {  
    // i está declarada en  
    línea,  
    // no existe fuera del  
    bucle.  
    // muestra 0, luego 1, luego  
    2  
    alert(i);  
}
```

- Podemos omitir cualquier parte del for:

```
for (; ;) {  
    // se repite sin límites  
}
```

- break y continue

El primero sale del bucle, el segundo salta a la siguiente iteración.

# Ejercicios

¿Cuál es el último valor mostrado por el alert?

```
let i = 3;
while (i) {
  alert(i--);
}
```

¿Qué se mostrará? ¿Se muestra lo mismo en los dos bucles?

- 1) 

```
let i = 0;
while (++i < 5) alert(i);
```
- 2) 

```
let i = 0;
while (i++ < 5) alert(i);
```

¿Qué muestra el for?

- 1) 

```
for (let i = 0; i < 5; i++) alert(i);
```
- 2) 

```
for (let i = 0; i < 5; ++i) alert(i);
```

# Funciones

Una declaración de función se ve así:

```
function  
name(parámetros,  
delimitados, por, coma)  
{  
    /* código */  
}
```

Los valores pasados a una función como parámetros se copian a sus variables locales.

Una función puede acceder a variables externas. Pero funciona solo de adentro hacia afuera. El código fuera de la función no ve sus variables locales.

Una función puede devolver un valor. Si no lo hace, entonces su resultado es undefined.



# Funciones

Para que el código sea limpio y fácil de entender, se recomienda utilizar principalmente variables y parámetros locales en la función, no variables externas.

Siempre es más fácil entender una función que obtiene parámetros, trabaja con ellos y devuelve un resultado, que una función que no obtiene parámetros, pero modifica las variables externas como un efecto secundario.

## Nomenclatura de funciones:

- Un nombre debe describir claramente lo que hace la función. Cuando vemos una llamada a la función en el código, un buen nombre nos da al instante una comprensión de lo que hace y devuelve.
- Una función es una acción, por lo que los nombres de las funciones suelen ser verbales.
- Existen muchos prefijos de funciones bien conocidos como create..., show..., get..., check... y así. Úsalos para insinuar lo que hace una función.

Las funciones son los principales bloques de construcción de los scripts. Ahora hemos cubierto los conceptos básicos, por lo que en realidad podemos comenzar a crearlos y usarlos. Pero ese es solo el comienzo del camino.