UD2: Estructura del lenguaje Javascript

Desarrollo Web en Entorno Cliente

Índice

Primeros pasos con JS

Indice

UD2: Estructura del lenguaje Javascript

Sintaxis básica

- Case-sensitive
  - Sensible a mayúsculas-minúsculas.
- Formato libre
  - Los saltos de línea y espacios en blanco no aportan significado.
- Fin de instrucción con punto y coma [;]
  - Opcional pero recomendable.
- **No se define el tipo de las variables**.

Javascript y HTML

- Un archivo JS siempre irá asociado a un archivo HTML que contiene la estructura de la página.
- Podemos incluir el código JS:
  - En el propio archivo HTML.
  - **En un archivo externo.**

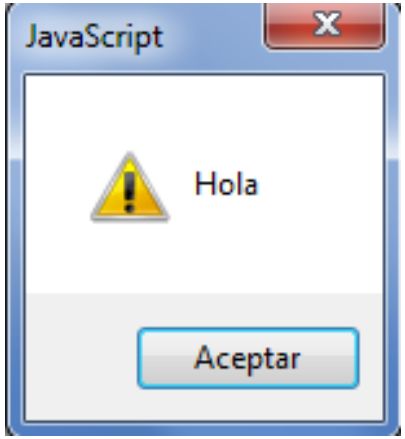
Javascript en el propio archivo

- Metemos el código dentro de la etiqueta `<script>`
  - Habitualmente la etiqueta se incluye en `<head>` pero puede ir también en `<body>`.
  - El código se cargará en la página en el momento en que añada la etiqueta.
    - Esto implica que no podré acceder a los elementos de la página si aún no se han cargado.
  - En HTML5 no es necesario poner el tipo MIME.
    - En versiones anteriores era necesario indicar el lenguaje:

```
<script type="text/javascript">
```

Javascript en el propio archivo: Ejemplo

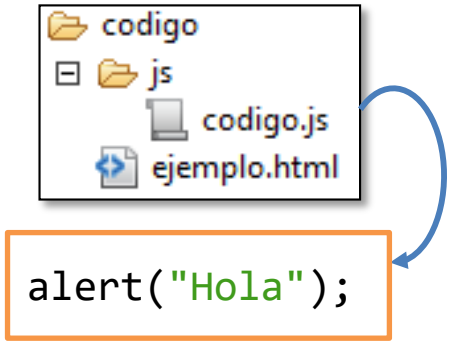
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
<title>Ejemplo</title>

    <script type="text/javascript">
    /* Lo que va aquí es código Javascript */
    alert("Hola");
    </script>

</head>
<body>
Aquí iría la página
</body>
</html>
```



Archivos externos

- Es la práctica más utilizada y recomendable.
- Definimos todo el código Javascript en un archivo externo (con extensión js).
  - Habitualmente introducimos los archivos en una carpeta diferenciada, por ejemplo `js` o `scripts`.
- Enlazamos con el archivo Javascript usando una etiqueta `<script>` e indicando la ruta del archivo.

```
<script src="js/codigo.js"></script>
```



<noscript>

- El contenido de la etiqueta se muestra si el navegador no permite Javascript

```
<noscript>
 Esta página no permite el uso de Javascript
</noscript>
```

CDATA

- Los analizadores XHMTL son intolerantes a elementos que no comienzan por < (como los usados en JS)
  - La solución es encapsular el código JS en una sección CDATA:

```
<script type="text/javascript">
//<![CDATA[

// código de JavaScript a continuación

//]]>
</script>
```

Comentarios

- Comentario de una línea:

```
//Imprimo un mensaje vital
alert('Propicios días');
```

- Comentario de varias líneas:

```
/*
 * Imprimo
 * un
 * mensaje
 * vital
 */
alert('Propicios días');
```

El intérprete los ignora pero se descargan con el resto del script

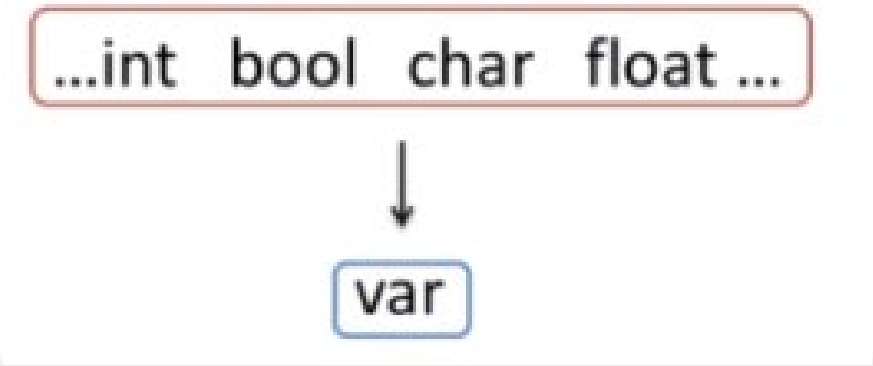
Variables

Indice

UD2: Estructura del lenguaje Javascript

Declaración de variables

- Elementos que uso para almacenar y hacer referencia a otros valores
- Anteponemos la palabra reservada **var**
  - No se declara el tipo de las variables
  - Puedo asignar un valor al declarar o posteriormente.
  - El tipo de una variable puede cambiar



No es obligatorio declarar las variables!
Pero si recomendable, pues las variables no declaradas son implícitamente globales

Variables: Ejemplos

```
var vidas=7;
/* Declaro la variable vidas e inserto en ella el valor numérico 7 */

var nombre="Player 1";
/* Declaro la variable nombre e inserto en ella una cadena de caracteres */

var seguirJugando=false;
/* Declaro la variable seguirJugado e inserto en ella un booleano */

var nombre;
/* Declaro una variable nombre pero no hago nada con ella */

nombre="Fermin";
/* Inserto un valor en nombre */

nombre="Juanillo";
/* Cambio el valor de nombre */

nombre=7;
/* Cambio el tipo de nombre, insertando otro tipo de datos */
```

Identificador

- Es el nombre de una variable.
- **Reglas**:
  - Sólo puede estar formado por letras, números, y los símbolos $ y _
  - El primer carácter no puede ser un número.
  - Se recomienda convención **camelCase para variables** y **mayúsculas para constantes** (variables cuyo valor no es susceptible de cambiar durante el programa).

```
var numero1;
var $numero1;

var letra
var $letra;
var $_letra;
```

```
var 1Nombre; /* Empieza por número */
var numero;1; /* Contiene ; */
```

Ámbito de las variables

GLOBAL

```
<script>
        var variableGlobal;

    function verAmbito() {
            var variableLocal;

        while(condicion){
                var varBloque;
        }

    }
</script>
```

LOCAL

BLOQUE

Declaración de constantes [ES6]

- Son variables cuyo valor no puede cambiar
- Se declaran con la palabra reservada **const**

```
const CAPITAL="Santander";
alert(CAPITAL+ " es la capital de Cantabria");

CAPITAL="Torrelavega";

alert(CAPITAL+ " es la capital de Cantabria");
```

Tipos de variables

- Dependen del valor que le asigno:
  - Numéricas (entero o decimal)

```
var partidas=99;
var fuerza=77.3;
```

  - Cadenas de texto (entre cadenas dobles o simples)

```
var mensajeBienvenida="Bienvenido Mr. Marshal";
var nombreJugador='El Fary';
var letraPulsada='c';
```

  - Booleanos [true/false]

```
var deseaContinuar=true;
var haPagado=false;
```

Comillas simples y dobles

```
/* El contenido de textol tiene comillas simples,
 * por eso se encierra entre comillas dobles */
var textol="Una frase con 'comillas simples' dentro";

/* El contenido de texto2 tiene comillas dobles,
 * por eso se encierra entre comillas simples */
var texto2='Una frase con "comillas dobles" dentro';
```

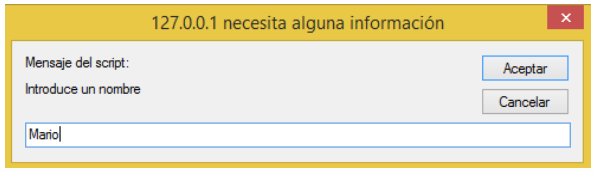
Mostrando y leyendo variables

- `alert(variable)`
  - Muestra el valor de la variable por en una ventana emergente
  - También admite un literal

```
var nombre="Mario";

alert(nombre);
alert ("Mario Bros");
```

- `variable=prompt(“Mensaje”)`
  - Muestra un formulario que permite dar valor a una variable

```
var nombre=prompt("Introduce un nombre");
```



Caracteres de escape

- Permiten introducir caracteres especiales en cadenas de texto

Si quiero incluir..	Debo introducir…
Una nueva línea	**\n**
Un tabulador	**\t**
Una comilla simple	**\’**
Una comilla doble	**\”**
Una barra invertida	****

```
var texto1='Una frase con \' comillas simples \' dentro';
var texto2="Una frase con \" comillas dobles \" dentro";
```

Arrays

- Es una colección de elementos que pueden ser del mismo o distinto tipo.

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves",
"Viernes", "Sábado", "Domingo"];
```

- Podemos acceder a los elementos del array a través de un índice
- Las posiciones de un array comienzan a contarse en 0 y no en 1

```
var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"
var otroDia = dias[5]; // otroDia = "Sábado"
```

Variables no inicializadas

- Tienen el valor `undefined`
  - No es deseable que las variables tengan dicho valor.
  - No confundir con `null`, que es el valor que asignamos a objetos que aún no han sido instanciados.

Conversión de tipos

- parseInt(cadena) ó parseFloat(cadena)
  - Obtiene un entero o flotante a partir de una cadena.

```
var numero=parseInt("30");
alert(numero);

var numero2=parseInt("40px");
alert(numero);

var numero3=parseInt("5pepe");
alert(numero);

var numero4=parseInt("8+2");
alert(numero);
```

30

40

5

8

```
var numero=parseFloat("3,2");
alert(numero);

var numero2=parseFloat("3.2");
alert(numero);
```

3

3.2

EJERCICIO PROPUESTO

- Crea un programa en el que crees 4 variables, 2 cadenas y 2 números, con los siguientes valores: tu nombre, tu apellido, tu edad y tu año de nacimiento.
- Muestra en un mensaje de alert tu nombre y apellidos separados por un salto de línea.
- Muestra en un mensaje de alert la suma de las variables edad y año de nacimiento.

EJERCICIO PROPUESTO

- Crea un programa en el que crees 4 variables, 2 cadenas y 2 números, y pida por prompt los siguientes valores: tu nombre, tu apellido, tu edad y tu año de nacimiento.
- Muestra en un mensaje de alert tu nombre y apellidos separados por un salto de línea.
- Muestra en un mensaje de alert la suma de las variables edad y año de nacimiento.

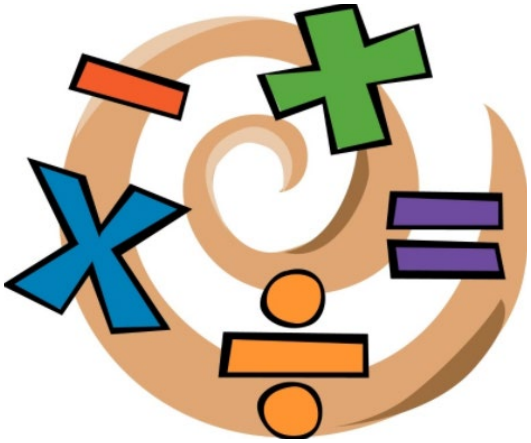
Operadores

Indice

UD2: Estructura del lenguaje Javascript

Operadores

- Por sí solas las variables no hacen gran cosa, necesitamos **operadores**.
- Permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar valores.
- Tipos:
  - Asignación.
  - Lógicos.
  - Matemáticos.
  - Relacionales.
  - Identificación de tipos



Asignación (=)

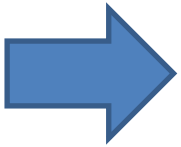
- Operador más utilizado y el más sencillo
- Se utiliza para guardar un valor en una variable.
- Dos partes:
  - Izquierda del igual: Nombre de variable.
  - Derecha del igual: Expresión (variables, valores, condiciones lógicas…)

```
var numero1=3;
var numero2=4;

numero1=5;
numero1=numero2;
```

Incremento (++) y decremento (--)

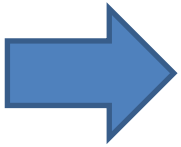
- Sólo se aplican a tipos numéricos

```
var numero=5;
numero=numero+1;

alert(numero); // numero=6
```



```
var numero=5;
++numero;

alert(numero) //numero=6
```

```
var numero=5;
numero=numero-1;

alert(numero); // numero=4
```



```
var numero=5;
--numero;

alert(numero) //numero=4
```

Inc. Y Dec.: Ejemplo

- Pueden ir
  - Delante de la variable (su valor se incrementa antes de ejecutar la sentencia donde aparece).
  - Después de la variable (su valor se incrementa después de ejecutar la sentencia donde aparece).

```
var numero1=5;
var numero2=2;

numero3=numero1++ + numero2;
```

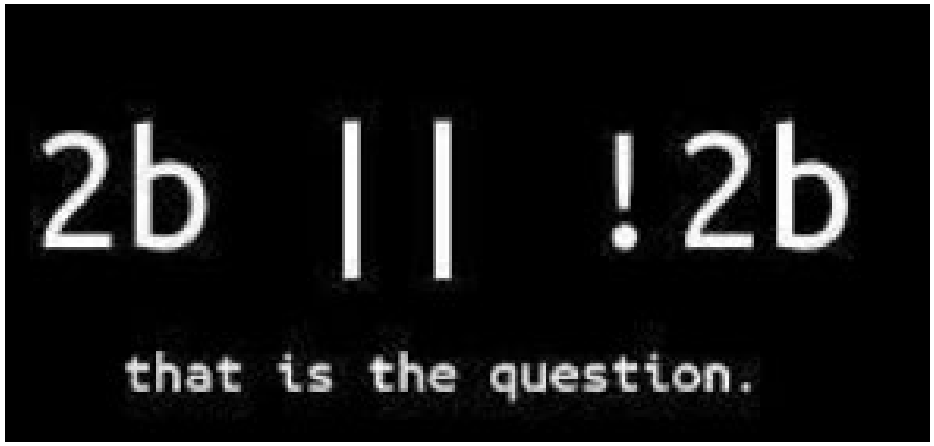
numero3=7, numero1=6

```
var numero1=5;
var numero2=2;

numero3= ++numero1 + numero2;
```

numero3=8, numero1=6

Operadores lógicos

- Devuelven un valor de tipo booleano.
- Realizan una operación lógica.
  - Negación (!).
  - "Y" Lógico(AND).
  - "O" Lógico (OR)



Operadores lógicos: Ejemplos

```
var valor1=true;
var valor2=false;

resultado=valor1 && valor2; //resultado=false

valor1=true;
valor2=true;

resultado=valor1 && valor2; //resultado=true
```

AND

```
var valor1=true;
var valor2=false;

resultado=valor1 || valor2; //resultado=true

valor1=false;
valor2=false;

resultado=valor1 || valor2; //resultado=false
```

OR

```
var visible=true;
alert(!visible); // "false"
```

NOT

Operadores matemáticos

- Realizan operaciones matemáticas elementales
  - Suma (+)
  - Resta (-)
  - Multiplicación (*)
  - División (/)
  - Resto de la división entera (%).
- Puedo combinarlos con el operador de asignación (+=, -=, *=, etc…).

Operadores matemáticos: Ejemplos

Operaciones elementales

```
var numero1=10;
var numero2=5;

resultado=numero1/numero2; //resultado=2
resultado=3 + numero1; //resultado=13
resultado=numero2-4; //resultado=1
resultado=numero1*numero2; //resultado=50
```

Resto

```
var numero1=10;
var numero2=5;

resultado=numero1%numero2; //resultado=0

numero1=9;
numero2=5;

resultado=numero1%numero2; //resultado=4
```

Operadores matemáticos: Ejemplos

Junto con operador de asignación

```
var numero1=5;

numero1+=3; //numero1=numero1+3;
numero1-=1; //numero1=numero1-1;
numero1*=2; //numero1=numero1*2;
numero1/=5; //numero1=numero1/5;
numero1%=4; //numero1=numero1%4;
```

Operador de suma (+)

- También se utiliza para concatenar arrays:

```
var frase1="Ojos que no ven";
var frase2=" corazón que no siente";

var refran1=frase1+frase2;
var refran2=frase1+" castañazo que te pegas";

alert(refran1);
alert(refran2);
```

Operadores relacionales

- Devuelven un valor booleano
  - Mayor que (>).
  - Menor que (<).
  - Mayor o igual (>=).
  - Menor o igual (<=).
  - Igual que (==)
  - Distinto de (!=).

Operadores relacionales: Ejemplos

```
var numero1=3;
var numero2=5;

resultado=numero1>numero2; //resultado=false;
resultado=numero1<numero2; //resultado=true;

numero1=5;
numero2=5;

resultado=numero1 >= numero2; //resultado=true;
resultado=numero1 <= numero2; //resultado=false;
resultado=numero1 == numero2; //resultado=true;
resultado=numero1 != numero2; //resultado=false;
```

Operadores relacionales (2)

- El operador *igual* (==) es origen de un gran número de errores de programación
- No confundir con el operador de asignación (=).

```
//El operador "=" asigna valores
var numero1=5;
resultado=numero1=3;
//numero1 vale 3 y resultado vale 3

//El operador "==" compara variables
var numero1=5;
resultado=numero1==3;
//numero1 vale 5 luego resultado vale false
```

Operadores relacionales (3)

- También pueden usarse para comparar cadenas de texto (alfabéticamente)

```
var texto1="Hola";
var texto2="Hola";
var texto3="Adios";

resultado=texto1==texto3;  //false
resultado=texto1!=texto2;  //false
resultado=texto1>=texto2;  //false
```

```
//¿Qué devolvería la siguiente sentencia?

resultado=texto1==texto2;
```

Comparación estricta

- === y !== permiten comparación **estricta**
  - Se compara el tipo de la variable además de su valor
  - Los operadores de comparación == y != son de comparación **relajada**
    - Si los tipos son distintos se trata de comparar para que sean comparables
    - La operación relajada tiene bastantes reglas que no siempre son fáciles de recordar, por eso por lo general se recomienda el uso de la **comparación estricta**.

Comparación estricta vs relajada

```
var numero=10;
var numero2="10";

if(numero==numero2){
    /* .... */
}
```

El **if** se evalúa como verdadero

```
var numero=10;
var numero2="10";

if(numero===numero2){
    /* .... */
}
```

El **if** se evalúa como falso

Identificador de tipos

- typeof
  - Devuelve una cadena que representa el tipo de dato contenido en una variable

```
var numero=25;
alert(typeof(numero));
```

number

```
var numero=2.5;
alert(typeof(numero));
```

number

```
var numero="25";
alert(typeof(numero));
```

string

```
var numero=false;
alert(typeof(numero));
```

boolean

Operador ternario

- Permite evaluar una condición y ejecutar una de dos condiciones en función de la misma.

```
condición ? expresión1 : expresión2;
```

- **Condición**: Expresión que podemos evaluar como verdadero o falso.
- **Expresión 1**: Se evalúa si condición es verdadero
- **Expresión 2**: Se evalúa si condición es falso.

```
var miEdad = 24;
var mayorEdad = (miEdad > 18) ? "Sí, eres mayor de edad" : "No, sigue intentando";
```

Estructuras de control

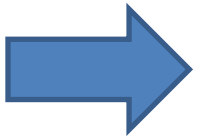
Indice

UD2: Estructura del lenguaje Javascript

Estructura condicional (`if`)

- Permite tomar decisiones en función del valor de una variable
- Las sentencias se ejecutan si se cumple la condición
  - Las llaves no son obligatorias si hay una única sentencia

```
if(condicion){
sentencias;
}
```



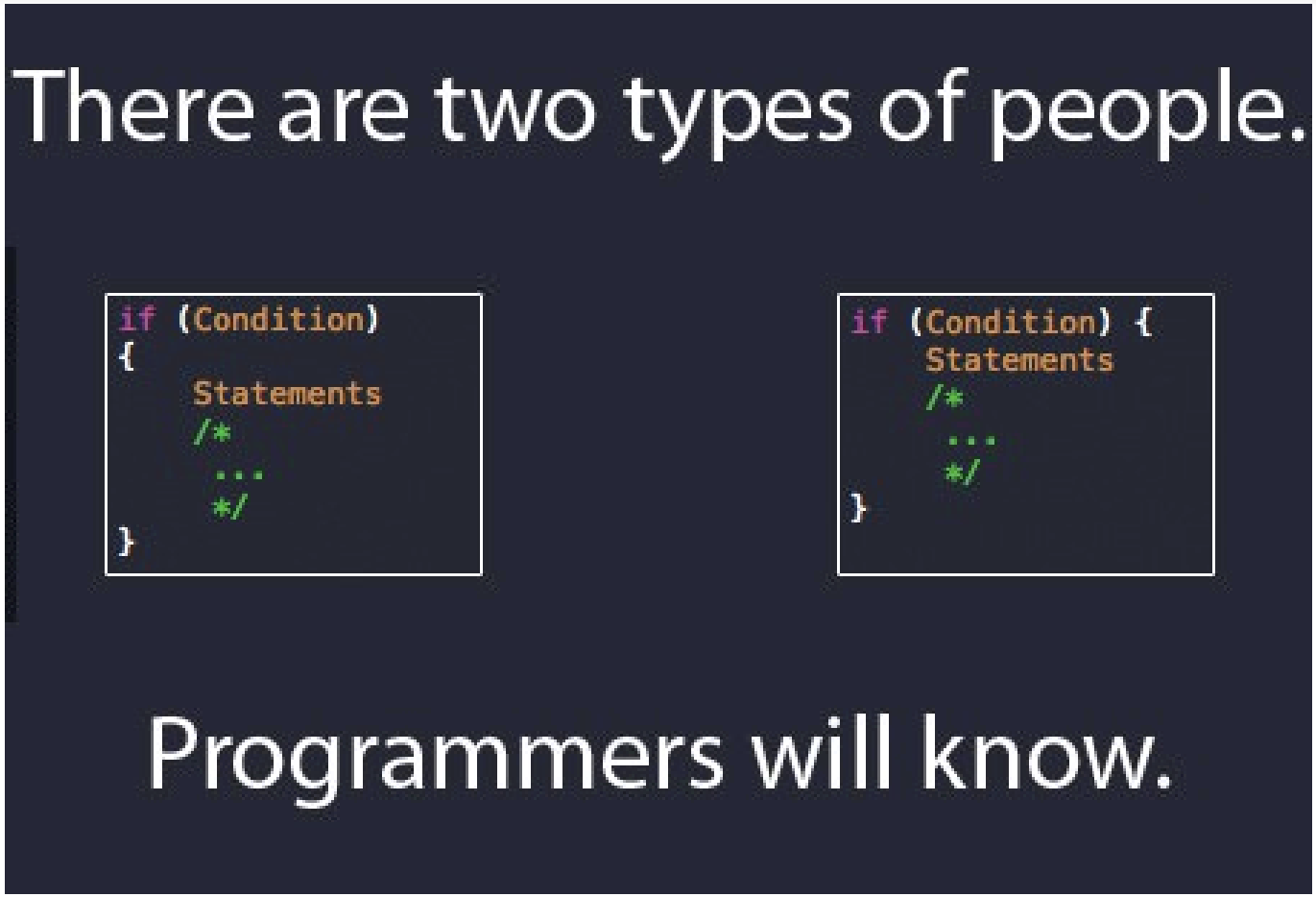
```
if(condicion==true){
sentencias;
}
```

Estructura condicional: Ejemplo

- ¿Se muestra el mensaje?

```
boolean seguirJugando=false;

if(seguirJugando=true){
alert("Sigues jugando");
}
```

Estructura condicional



Estructura condicional (if) (2)

- Puedo encadenar varias condiciones simples o complejas.

```
var mostrado=false;
if(!mostrado)
  alert("ZAS!");
```

```
var mostrado=false;
var usuarioPermiteMensajes=true;

if(!mostrado && usuarioPermiteMensajes)
  alert("PUM!");
```

Estructura condicional alternativa (`if..else`)

- Permite decir qué acciones ejecutamos si no se cumple la condición del **`if`**.

```
if(condicion){
   sentencias;
}
else{
   otrasSentencias;
}
```

- Los paréntesis habitualmente se usan si hay más de una sentencia.
- Puedo encadenar el **`else`** con un nuevo **`if`**.

Estructura condicional alternativa: Ejemplos

```
var edad=18;
if(edad>=18){
 alert("Felicidades, ya eres mayor de edad");
}
else{
 alert("Todavía eres menor de edad, pringao");
}
```

```
if(edad<=12)
 alert("Eres un niño")
else if (edad<19)
 alert("Eres adolescente");
else if(edad<35)
 alert("Sigues siendo joven");
else
 alert("Piensa en cuidarte un poco más");
```

Estructura condicional: Ejemplo

- Completar las condiciones de los **`if`** del siguiente script para que los mensajes de los **`alert()`** se muestren siempre de forma correcta:

```
var numero1=parseInt(prompt("Introduzca un número"));
var numero2=parseInt(prompt("Introduzca un número"));

if(...)
 alert("numero1 no es mayor que numero2");
if(...)
 alert("numero2 es positivo");
if(...)
 alert("numero1 es negativo o distinto de cero");
if(...)
 alert("Incrementar en 1 unidad numero1 no lo hace mayor o igual que
numero1");
```

Estructura condicional: Ejemplo

```
var numero1=parseInt(prompt("Introduzca un número"));
var numero2=parseInt(prompt("Introduzca un número"));

if(!(numero1>numero2))
  alert("numero1 no es mayor que numero2");
if(numero2>0)
  alert("numero2 es positivo");
if((numero1<)||(numero1!=0))
  alert("numero1 es negativo o distinto de cero");
 if(++numero1<numero2)
  alert("Incrementar en 1 unidad numero1 no lo hace mayor o igual que
numero1");
```

Estructura múltiple (switch)

- Se basa en evaluar una expresión con resultado escalar, para decidir el punto de entrada en la estructura.

```
switch(expresion){
 case valor1:
      sentencia1;
       break;

 case valor2:
      sentencia2;
       break;

 case valor3:
      sentencia3;
       break;
}
```

Bucle for

- Permite repetir una o varias sentencias un número determinado de veces

```
for(inicialiación;condición;actualización){
 sentencias;
}
```

  - **inicialización**: Valores iniciales de la variable que controla la repetición.
  - **condición**: Que debe cumplirse para que las sentencias se ejecuten.
  - **actualización**: Se ejecutan después de cada repetición. Normalmente actualiza la variable que controla la repetición.

Bucle for: Ejemplos

```
var mensaje="Yuhuuu! Estoy dentro de un bucle!!";

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

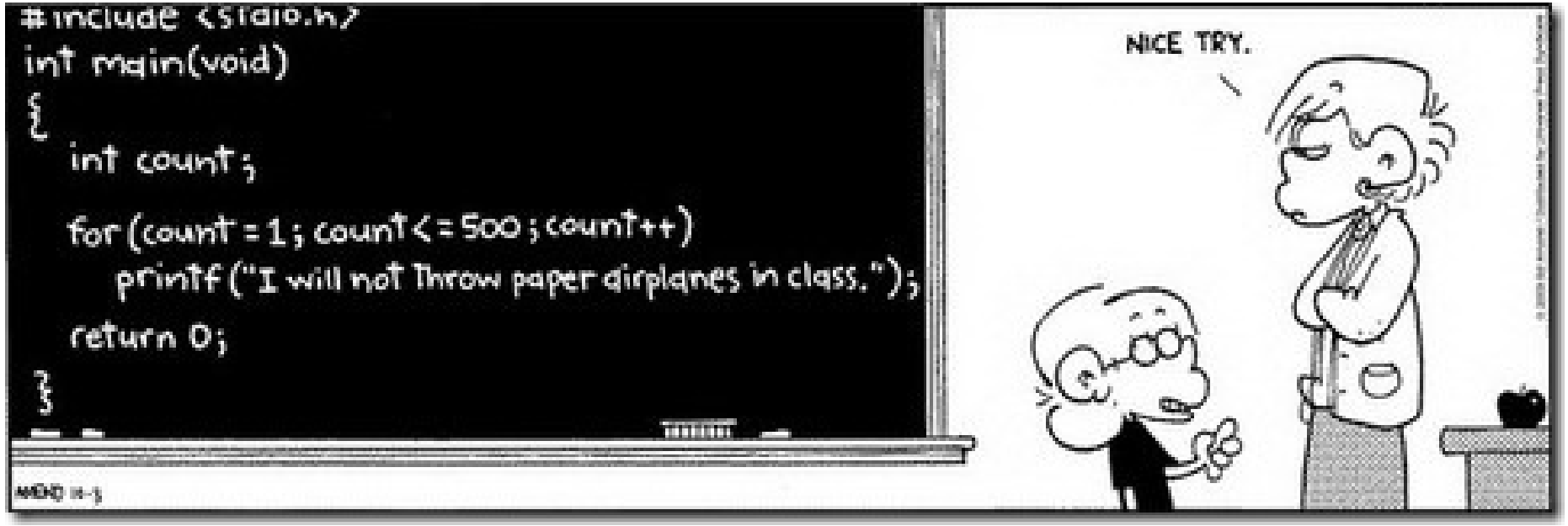
Inicialización

Condición

Actualización

```
var dias=["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
"Sabado", "Domingo"];

for(var i=0; i<7; i++)
 alert(dias[i]);
```

Bucles for: ¿Para qué valen?



Bucle for…in

- Se utiliza para recorrer todos los elementos que forman un array.
  - Funciona aunque cambie el número de elementos del array.

```
var dias=["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
"Sabado", "Domingo"];

for(i in dias)
 alert(dias[i]);
```

Bucle `while`

- Repite una serie de sentencias mientras se repita una condición.
- La condición no podría llegar a ejecutarse nunca.
- Habitualmente en las sentencias controlo la condición de salida del bucle.

```
while(condición){
sentencias;
}
```

Bucle `while`: Ejemplo

- ¿Qué hace el siguiente script?

```
var resultado=0;
var numero=100;
var i=0;

while(i<=numero){
resultado+=i;
i++;
}

alert(resultado);
```

Bucle `do..while`

- Repite una serie de sentencias mientras se repita una condición
  - La diferencia con el **`while`** es que las sentencias se ejecutan **al menos una vez**, mientras que con el while podrían no ejecutarse nunca.

```
do{
 sentencias;
} while(condición);
```

Bucle do…while: Ejemplo

- ¿Qué hace el siguiente script?

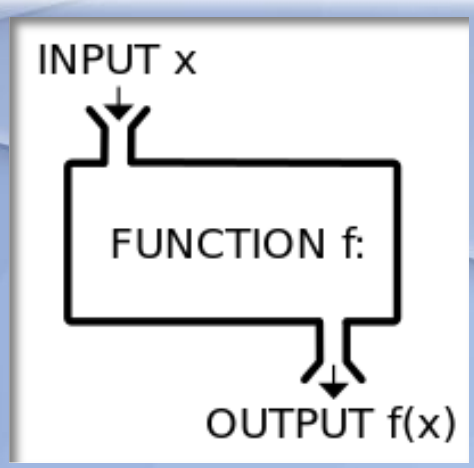
```
var resultado=1;
var numero=5;
do{
 resultado*=numero;
 numero--;
} while(numero>0);
alert(resultado);
```

EJERCICIO PROPUESTO

- Define un array numérico formado por números del 1 al 9.
- El script debería comprobar si hay algún número fuera de su posición, informando SOLO del primer caso que se encuentre.
  - Por ejemplo, en el array [1,2,3,9,5,6,7,8,9] se debería obtener el mensaje "Error en la posición 3, hay un 9 cuando se esperaba un 4".
- En caso de que todos los elementos estén en la posición correcta, se mostrará un mensaje indicativo.
- Los mensajes se mostrarán por alert.

EJERCICIO PROPUESTO

- Realiza una nueva versión del script anterior donde se detecten TODOS los números que estén fuera del orden esperado.
- Al igual que en el caso anterior, si todos los elementos están en la posición correcta se mostrará un mensaje indicativo.
- Los mensajes se mostrarán por alert.



Funciones simples [Adelanto]

Indice

UD2: Estructura del lenguaje Javascript

Funciones

- Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y se pueden reutilizar de manera sencilla.
- Facilitan mucho la organización, y por consiguiente el mantenimiento y depuración de los programas.
- Se suele diferenciar:
  - **Procedimientos**: Sólo ejecutan acciones
    - No existen en JS.
  - **Funciones**: Ejecutan acciones y devuelven valores

Funciones simples

- Primero declaramos la función y luego la utilizamos (llamada o invocación):

```
/* Definición */
function nombreFuncion(){
sentencias;
}

/* Llamada o invocación */
nombreFuncion();
```

Funciones simples: Ejemplo

```
function sumayMuestra(){
 var resultado = numero1 + numero2;
 alert("El resultado es "+ resultado);
}
```

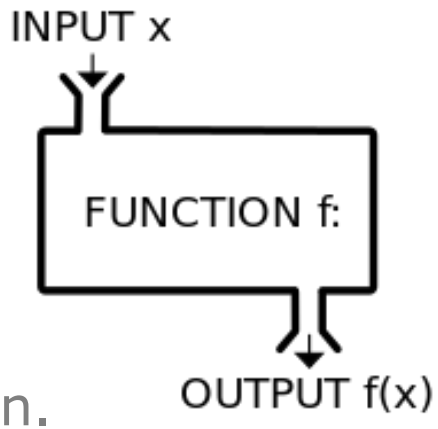
Definición

```
var resultado;
var numero1=3;
var numero2=5;

sumayMuestra();

numero1=5;
numero2=6;
sumayMuestra();
```

Llamada o invocación

E/S de datos en funciones

- **Argumentos/parámetros**
  - Permiten especificar las **entradas** de la función
  - Ejemplo (suma): Los sumandos.
- **Retorno**
  - Especifica el **valor que devuelve** la función.
  - Ejemplo (suma): El resultado de la suma.




```
/* Definición */
function nombreFuncion(argumento1, argumento2){
sentencias;
return valor;
}
```

Argumentos: Ejemplo

```
function sumayMuestra(primerNumero,segundoNumero){
 var resultado = primerNumero + segundoNumero;
 alert("El resultado es: "+ resultado);
}
```

Definición

```
//Declaración de las variables
var numero1=3;
var numero2=5;

//Llamada a la función
sumayMuestra(numero1,numero2);
```

Valor de retorno: Ejemplo

```
function suma(primerNumero,segundoNumero){
 var resultado = primerNumero + segundoNumero;
 return resultado;
}
```

Definición

```
//Declaración de las variables
var numero1=3;
var numero2=5;

//Llamada a la función
var resultado=suma(numero1,numero2);
alert(resultado);
```

EJERCICIO PROPUESTO

- Realiza un script que pida un número entero por pantalla y a continuación genere sus tablas de multiplicar, sumar y dividir del 1 al 10 (es decir, el resultado de multiplicar, sumar y dividir el número indicado sucesivamente por los números del 1 al 10.
- El resultado se presentará en 3 mensajes por consola (uno para cada operación).
- Resuelve el ejercicio las siguientes funciones:
  - suma(entero, entero): entero.
  - multiplica(entero,entero): entero
  - divide(entero,entero):flotante
  - tablaMultiplicar(entero):cadena
  - tablaSumar(entero):cadena
  - tablaDividir(entero):cadena

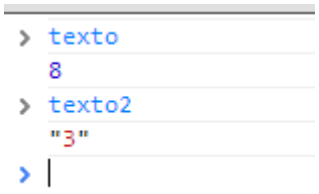
Utilidades

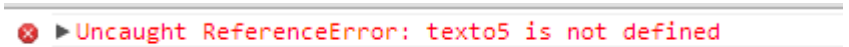
Indice

UD2: Estructura del lenguaje Javascript

Herramientas de depuración

- Los principales navegadores incluyen herramientas de depuración de Javascript que permiten realizar tareas en tiempo de ejcución:
  - Visualizar errores y avisos en tiempo.
  - Inspeccionar el valor de las variables
- El manejo correcto de estas herramientas es imprescindible para el desarrollo en Javascript.





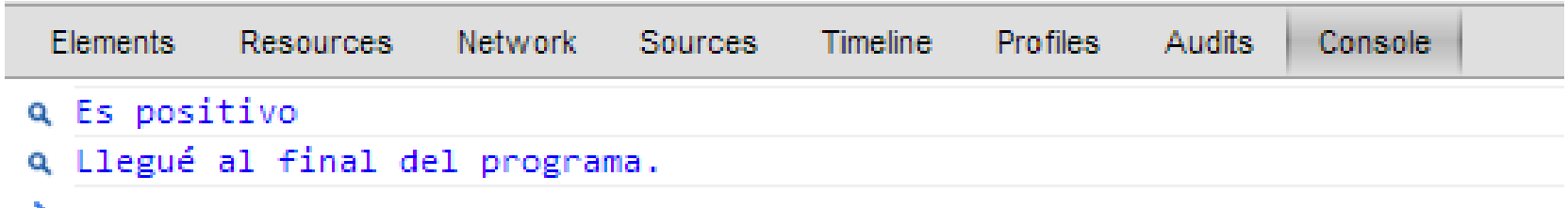
Objeto console

- Ofrece una interfaz para comunicarnos por código con la consola
  - No es estándar pero viene incluida en la mayor parte de los navegadores.

```
/* AVISOS (WARNING) */
console.warn("Mensaje");

/* ERRORES */
console.error("Mensaje");

/* INFORMACIÓN (LOG) */
console.log("Mensaje");

/* INFORMACIÓN (INFO) */
console.info("Mensaje");

/* DEPURACIÓN (DEBUG) */
console.debug("Mensaje");
```

Objeto console : Ejemplo

```
var numero=parseInt(prompt("Introduzca un número positivo"));

if(numero>0){
 console.log("Es positivo");
 hazCosas();
}
else{
 console.log("Es negativo");
 hazOtrasCosas();
}

console.log("Fin del programa");
```



Ofuscadores de código

- Al ejecutarse en cliente cualquiera que visualice la página puede ver el contenido de un script
- Los ofuscadores de código hacen que el código sea difícil de interpretar por humanos sin perder funcionalidad
  - Elimina saltos de línea, espacios en blanco, cambio de nombre de variables.