



# Introducción al Lenguaje de JavaScript.

---

## Unidad Trabajo nº 2

Autor: David García Fernández



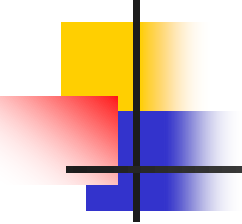
Este obra está bajo una [licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).



## Objetivos de la Unidad Trabajo

---

- Conocer las bases sintácticas del lenguaje JavaScript.
- Saber programar un script sencillo empleando la sintaxis básica del lenguaje.
- Saber usar variables en el código, así como realizar conversiones y operaciones básicas.
- Emplear bloques de código y estructuras repetitivas.



# Fundamentos de JavaScript. Introducción.

---

- Brendan Eich(1961-): Desarrolla JavaScript en 1995 para Netscape con el objetivo de dinamizar la web.
- Críticas (lento, inseguro, incompatibilidades) y estandarización (1996): ECMAScript o ECMA-262
- Dialectos: JavaScript, JScript, ActionScript, ...
- Hitos en el desarrollo de JavaScript:
  - Inicios (1996-1997): Validación de formularios y rollovers
  - DOM (1997-2004): Llega la animación
  - Ajax (1999-): La hora de las aplicaciones web
  - Bibliotecas (2005-): Hay que mejorar la productividad
  - HTML5 (2008-): Pilar esencial de tecnologías revolucionarias: Canvas (\*), Audio y vídeo (\*), Arrastrar y soltar (\*), Aplicaciones offline, Almacenamiento web, Geolocalización, Web Workers (\*), Notificaciones (\*)



# Fundamentos de Javascript.

---

- Normas sintácticas básicas:
  - Sensible al uso de **mayúsculas/minúsculas**
  - Recomendaciones:
    - Nombres de variables/funciones/objetos en **minúscula**
    - Usar **lowerCamelCase**
- Comentarios:
  - Una línea: //
  - Varias líneas: /\* ... \*/
- Cada instrucción debe terminar con un **punto y coma ;**
- Los **espacios consecutivos** son considerados como un único espacio
  - Recomendaciones:
    - Usar sangrados de 4 espacios
    - No crear líneas con más de 80 caracteres (partir con saltos de línea).



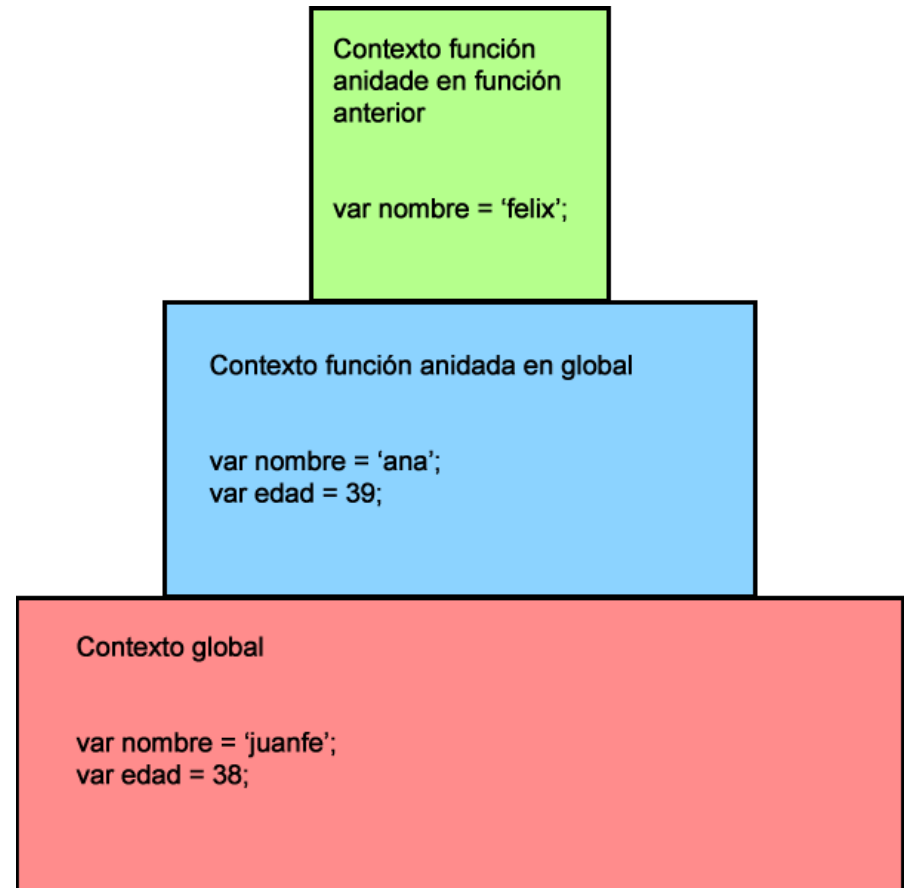
# Fundamentos de Javascript.

---

- Variables y tipos de datos
  - **Tipado débil:** el tipo de datos no es una propiedad de las variables, sino de los datos.
  - No es obligatorio declarar las variables, pero puede hacerse con **var**.
- Alcance de las variables
  - Una variable no declarada se considera global.
  - Una variable declarada con var dentro de una función es local dentro de esa función.
  - Una variable declarada con var fuera de una función se considera global.
  - Si una función posee una variable local cuyo nombre coincide con el de otra variable global, dentro de la función se utiliza la local.

# Fundamentos de Javascript.

- Contextos de ejecución
  - Conjunto de variables declaradas dentro de una función.
- Cadena de alcance
  - Pila de contextos de ejecución cargados en cada instante de la ejecución.
  - JavaScript busca las variables desde el contexto más alto de la pila hacia el global.





## Ejercicio 1.

---

- Crea un fichero html en el que “incrustes” el siguiente contenido:

```
<script type="text/javascript">  
    var primer_saludo = "Hola";  
    var segundo_saludo = primer_saludo;  
    primer_saludo = "Buenas...";  
    alert (segundo_saludo);  
</script>
```
- Antes de ver el resultado ¿Cuál cree que será el valor de la ventana emergente?



# Uso de var y let

---

- Las sentencias var y let permiten declarar y definir variables en javascript.
- let fue definido en el ECMA215 (ES6) permitiendo declarar las variables con operatividad a nivel de bloque {}
- Las variables definidas con let no pueden ser redeclaradas.
- Con let no pueden ser usadas sin haberse declarado.

```
{ let a = 1 }  
// la variable a NO se puede usar fuera del bloque.  
  
{ var b = 1 }  
// la variable a SÍ se puede usar fuera del bloque.
```





# Fundamentos de Javascript.

## ■ Tipos de datos en JavaScript:

### ■ **number:**

- Números en notación decimal (12.56), hexadecimal (0xA1F2) o científica (1256e-2).
- Comprendidos entre Number.MAX\_VALUE y Number.MIN\_VALUE.
- $0.1 + 0.2 == 0.3$  se evalúa como false porque 0.1 no tiene una representación exacta en IEEE 754.

### ■ **string**

- Cadena de caracteres entre comillas dobles o simples indiferentemente.
- Escapado de caracteres: \', \", \\ y \n
- `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />`
- Unicode 16 bits: \unnnn (<http://unicode.org/charts>)

á	é	í	ó	ú	Á	É	Í
\u00e1	\u00e9	\u00ed	\u00f3	\u00fa	\u00c1	\u00c9	\u00cd
Ó	Ú	ü	Ü	ñ	Ñ	ı	¿
\u00d3	\u00da	\u00fc	\u00dc	\u00f1	\u00d1	\u00a1	\u00bf



# Fundamentos de Javascript.

---

- **boolean:** true y false
- **null:** Sólo admite el dato null.
  - Es la respuesta a un convenio de los programadores que querían un tipo de datos para las variables que van a contener objetos pero aún no han sido asignadas.
  - No se puede aplicar implícitamente porque JavaScript no sabe qué va a contener en el futuro una variable.
- **undefined:** Sólo admite el dato undefined
  - Se aplica implícitamente a todas las variables declaradas pero no asignadas.
- **object**
  - Colección de datos, sin restricción en cuanto a tipo, identificados cada uno por un nombre.
  - `var automovil = new Object();`
  - Métodos y propiedades
    - Método: `automovil.repostar = function() {combustibleDisponible++};`
    - Propiedad: `automovil.combustibleDisponible = 30;`



# Fundamentos de Javascript.

---

- La asignación de datos con el signo igual siempre es por **valor**, excepto en los datos de tipo object, que es por **referencia**.

```
var colores = new Object();  
colores.favorito = 'azul';  
var miscolores = colores;  
miscolores.favorito = 'rojo';  
alert(colores.favorito);
```

- JavaScript posee una serie de objetos predefinidos, denominados objetos del núcleo, como: Number, String, Boolean, Array, Function, Date y Math.



# Fundamentos de Javascript.

---

- Cuando en una expresión interviene un number, string o boolean se crea automáticamente para él un objeto de tipo Number, String o Boolean, respectivamente, de modo que podemos acceder a sus propiedades.

```
var clave = "password";  
alert (clave.length);
```

- Otros tipos de objetos disponibles en JavaScript:
  - DOM: document
  - BOM: window

# Fundamentos de Javascript.

## Categoría de Operadores.

Categorías de operadores en JavaScript	
Tipo	Qué realizan
<b>Comparación.</b>	Comparan los valores de 2 operandos, devolviendo un resultado de true o false (se usan extensivamente en sentencias condicionales como <code>if...</code> <code>else</code> y en instrucciones <code>loop</code> ). == != === !== > >= < <=
<b>Aritméticos.</b>	Unen dos operandos para producir un único valor que es el resultado de una operación aritmética u otra operación sobre ambos operandos. + - * / % ++ -- +valor -valor
<b>Asignación.</b>	Asigna el valor a la derecha de la expresión a la variable que está a la izquierda. = += -= *= /= %= <<= >= >>= >>>= &=  = ^= []
<b>Boolean.</b>	Realizan operaciones booleanas aritméticas sobre uno o dos operandos booleanos. &&    !
<b>Bit a Bit.</b>	Realizan operaciones aritméticas o de desplazamiento de columna en las representaciones binarias de dos operandos. &   ^ ~ << >> >>>
<b>Objeto.</b>	Ayudan a los scripts a evaluar la herencia y capacidades de un objeto particular antes de que tengamos que invocar al objeto y sus propiedades o métodos. . [] () delete in instanceof new this
<b>Misceláneos.</b>	Operadores que tienen un comportamiento especial. , ?: typeof void

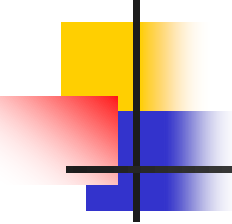


# Fundamentos de Javascript.

## Operadores Aritméticos.

---

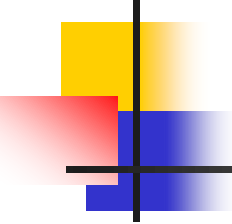
Sintaxis	Nombre	Tipos de Operando	Resultados
+	Más.	integer, float, string.	integer, float, string.
-	Menos.	integer, float.	integer, float.
*	Multiplicación.	integer, float.	integer, float.
/	División.	integer, float.	integer, float.
%	Módulo.	integer, float.	integer, float.
++	Incremento.	integer, float.	integer, float.
--	Decremento.	integer, float.	integer, float.
+valor	Positivo.	integer, float, string.	integer, float.
-valor	Negativo.	integer, float, string.	integer, float.



# Fundamentos de Javascript.

## Operadores de comparación.

Sintaxis	Nombre	Tipos de operandos	Resultados
==	Igualdad.	Todos.	Boolean.
!=	Distinto.	Todos.	Boolean.
===	Igualdad estricta.	Todos.	Boolean.
!==	Desigualdad estricta.	Todos.	Boolean.
>	Mayor que .	Todos.	Boolean.
>=	Mayor o igual que.	Todos.	Boolean.
<	Menor que.	Todos.	Boolean.
<=	Menor o igual que.	Todos.	Boolean.



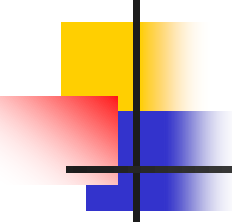
# Fundamentos de Javascript.

## Operadores de asignación.

---

Sintaxis	Nombre	Ejemplo	Significado
=	Asignación.	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	Sumar un valor.	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	Substraer un valor.	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	Multiplicar un valor.	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	Dividir un valor.	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	Módulo de un valor.	<code>x %= y</code>	<code>x = x % y</code>
<code>&lt;&lt;=</code>	Desplazar bits a la izquierda.	<code>x &lt;&lt;= y</code>	<code>x = x &lt;&lt; y</code>

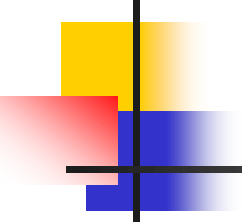




# Fundamentos de Javascript.

## Operadores de asignación.

Sintaxis	Nombre	Ejemplo	Significado
<code>&gt;=</code>	Desplazar bits a la derecha.	<code>x &gt;= y</code>	<code>x = x &gt; y</code>
<code>&gt;&gt;=</code>	Desplazar bits a la derecha rellenando con 0.	<code>x &gt;&gt;= y</code>	<code>x = x &gt;&gt; y</code>
<code>&gt;&gt;&gt;=</code>	Desplazar bits a la derecha.	<code>x &gt;&gt;&gt;= y</code>	<code>x = x &gt;&gt;&gt; y</code>
<code>&amp;=</code>	Operación AND bit a bit.	<code>x &amp;= y</code>	<code>x = x &amp; y</code>
<code> =</code>	Operación OR bit a bit.	<code>x  = y</code>	<code>x = x   y</code>
<code>^=</code>	Operación XOR bit a bit.	<code>x ^= y</code>	<code>x = x ^ y</code>
<code>[]=</code>	Desestructurando asignaciones.	<code>[a,b]=[c,d]</code>	<code>a=c, b=d</code>



# Fundamentos de Javascript.

## Operadores booleanos.

---

Sintaxis	Nombre	Operandos	Resultados
&&	AND.	Boolean.	Boolean.
	OR.	Boolean.	Boolean.
!	Not.	Boolean.	Boolean.

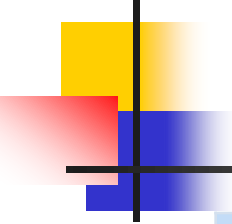


# Fundamentos de Javascript.

## Operadores bit a bit.

---

Operador	Nombre	Operando izquierdo	Operando derecho
&	Desplazamiento AND.	Valor integer.	Valor integer.
	Desplazamiento OR.	Valor integer.	Valor integer.
^	Desplazamiento XOR.	Valor integer.	Valor integer.
~	Desplazamiento NOT.	(Ninguno).	Valor integer.
<<	Desplazamiento a la izquierda.	Valor integer.	Cantidad a desplazar.
>>	Desplazamiento a la derecha.	Valor integer.	Cantidad a desplazar.
>>>	Desplazamiento derecha rellenando con 0.	Valor integer.	Cantidad a desplazar.



# Fundamentos de Javascript.

## Operadores de objeto.

Operador	descripción	Ejemplo
. (punto)	indica que el objeto a su izquierda tiene o contiene el recurso a su derecha	objeto.propiedad y objeto.método()
[]	Enumerar miembros de un objeto	var a =["Santiago","Coruña", "Lugo"]
delete	Eliminar un elemento de una colección.	delete oceanos[2]
in	Inspeccionar métodos o propiedades de un objeto.	"write" in document
instanceOf	comprobar si un objeto es una instancia de un objeto nativo de JavaScript	a instanceof Array;
new	Crear un objeto	var s = new String('rafa')
this	Referencia a un objeto.	var valorDelInput = this.value;



# Fundamentos de Javascript.

## Operadores misceláneos.

---

Operador	descripción	Ejemplo
,	indica una serie de expresiones que van a ser evaluadas en secuencia	var nombre, direccion, apellidos, edad;
typeof	Devuelve el tipo de dato de una variable o expresión.	
?:	Evalúa una condición	var h = a > b ? a : b;
void	No devuelve valor la expresión o función	void expresión



## Ejercicios 2.

---

- Ejecuta el siguiente código embebido en un html:

```
<script type="text/javascript">  
    var dividendo = prompt ("Introduce el dividendo: ");  
    var divisor = prompt ("Introduce el divisor: ");  
    var resultado;  
    divisor !=0 ? resultado = dividendo/divisor :  
                alert("No es posible la división por cero");  
    alert ("El resultado es: " + resultado);  
</script>
```

- Identifica los distintos tipos de operadores que has utilizado, de qué tipo son y analiza su función.



# Sentencias de control de ejecución.

---

- En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control".
- Las estructuras de control están "gobernada" por una expresión que es analizada como "condición" y que es evaluada como true o false.
- Tipos:
  - Condicionales:
    - ?:
    - if... else
    - switch ... case
  - Bucles de repetición:
    - for
    - while
    - do... while
    - Elementos de ruptura de bucle o iteración: break y continue.



# Sentencias de control de ejecución.

- **if .... else:**

La expresión de control es evaluada para ejecutar las instrucciones que delimitan cada bloque (true o false).

```
if (expresiónDeControl){  
    ...instrucciones a ejecutar si la expresión de control es verdadera...  
} else {  
    ...instrucciones a ejecutar si la expresión de control es falsa...  
}
```

- En el caso de querer evaluar solo si la expresión es verdadera, no se utiliza el bloque else:

```
if (expresiónDeControl){  
    ...instrucciones a ejecutar si la expresión de control es verdadera...  
}
```

- Ejemplo:

```
var edad = 37;  
if (edad % 2 ==0) {  
    alert('La edad es par');  
} else {  
    alert('La edad es impar');  
}
```





# Sentencias de control de ejecución.

---

- **? :**

JavaScript también permite utilizar asignaciones a variables en base a una condición.

*variable = (expresión de condición) ? valor1 : value2*

Ejemplo:

```
var h = a > b ? a : b;
```



# Sentencias de control de ejecución.

---

- **switch .... case:**

La expresión de control es evaluada entre diferentes valores definidos por cada case, cuando encuentra una correspondencia verdadera (true) se ejecutan las sentencias contenidas en el bloque dicho bloque case.

```
switch (expresiónDeControl){  
    case valor1:  
        ...bloque de instrucciones a ejecutar si expresiónDeControl coincide con valor1...  
        break;  
    case valor2:  
        ...bloque de instrucciones a ejecutar si expresiónDeControl coincide con valor1...  
        break;  
    ...  
    default:  
        ...bloque de instrucciones a ejecutar si expresiónDeControl no coincide con ningún caso...  
}
```



# Sentencias de control de ejecución.

- **switch .... case:**

Ejemplo:

```
var fabricante = 'citroen';
switch (fabricante){
    case 'ford':
        alert('Talleres Velasco');
        break;
    case 'citroen':
        alert('Talleres Doctor Esquerdo');
        break;
    case 'peugeot':
        alert('Talleres Las Acacias');
        break;
    default:
        alert('No existen talleres concertados para ese fabricante');
}
```



# Sentencias de control de ejecución.

- **for:**

- Se ejecuta repetidamente hasta que la expresión de control es falsa.
- Para ello es necesario que se inicialice el bucle a través de una instrucción de inicio (suele ser una variable de control de la expresión a evaluar).
- Al final de cada iteración se ejecuta una instrucción que modifica la expresión a evaluar. (normalmente incrementando o decrementando la variable evaluada en la expresión de control)

*for( instrucciónDeIniciación; expresiónDeControl; instrucciónTrasCadaIteración){  
...bloque de instrucciones a ejecutar en cada iteración...  
}*

- Ejemplo:

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



# Sentencias de control de ejecución.

---

- **while:**

- Se ejecuta repetidamente hasta que la expresión de control es falsa.
- Conlleva cambiar dentro del bloque de repetición la variable de control evaluada.

```
while (expresiónDeControl){  
    ...bloque de instrucciones a ejecutar en cada iteración...  
}
```

- Ejemplo:

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



# Sentencias de control de ejecución.

- **do...while:**

- Se ejecuta repetidamente hasta que la expresión de control es falsa.
- Siempre se ejecuta al menos una iteración ya que la expresión de control es evaluada después de las instrucciones del bloque.
- Conlleva cambiar dentro del bloque de repetición la variable de control evaluada.

```
do{  
    ...bloque de instrucciones a ejecutar en cada iteración...  
}while (expresiónDeControl)
```

- Ejemplo:

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



# Sentencias de control de ejecución.

---

- Existen otras formas de bucles para el manejo de array o elementos enumerados:

- for in

```
for (clave in object) {  
    // code block to be executed  
}
```

- for of

```
for (variable of iterable) {  
    // code block to be executed  
}
```

- Método `forEach()` del objeto `Array`.

```
array.forEach()
```

Nota: Estas formas de bucle se estudiarán más detalladamente en la unidad de Objetos en Javascript.



# Depuración del código y mensajes por consola.

---

- Los IDE disponen de herramientas y utilidades de depuración para detectar posibles errores y optimizar el código.
- Los navegadores también disponen de herramientas para desarrolladores que permiten la depuración.
- Javascript ofrece el objeto **console** para acceder a la consola de depuración del navegador.
- Se pueden usar métodos de console que visualicen mensajes exclusivamente en la consola y no en la página web HTML.
  - `console.log();`
  - `console.info();`
  - `console.warn();`
  - `console.error();`





## Ejercicios 3.

---

- Escriba un script que analice el valor de una variable llamada numero (que contiene un número entero positivo mayor que 1) y nos indique si se trata de un número primo o no.
- Utiliza métodos de console para visualizar mensajes que puedan ayudarte a seguir la ejecución del código.



# Salida en pantalla en Javascript.

---

Las formas de enviar mensajes a pantalla vistas hasta ahora han sido:

- `alert()` -> Función propia del elemento window.
- Métodos de console -> utilizados para mandar mensajes de error, avisos, información y logs.

Y además, existen:

- Utilizar la propiedad de `innerHTML` donde se define el contenido de un elemento HTML concreto.
- Utilizar el método `write()` del elemento document.
  - Esta forma reescribe el documento HTML, lo que implica que pueden eliminarse partes del HTML que se hayan escrito previamente.



# Salida en pantalla en Javascript.

---

## Ejemplo de innerHTML:

```
<!DOCTYPE html>
<html>

  <h1> Ejemplo de innerHTML </h1>

  <p id="ejemplo"> Valor inicial </p>

  <script>
    document.getElementById("ejemplo").innerHTML="Cambio de párrafo"
  </script>
</html>
```



# Salida en pantalla en Javascript.

---

Ejemplo de document.write():

```
<!DOCTYPE html >
<html>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <h1> Ejemplo documnet.write</h1>

  <p id="ejemplo"> Valor inicial </p>

  <button type="button" onclick="document.write('Cambio de párrafo')"> Probar </button>

</html>
```



# Prácticas 2.

---

## ■ EJERCICIO 1.

- Crea un script que pida introducir tres números y que los ordene de mayor a menor.
- Realiza las validaciones pertinentes evitando datos no numéricos o vacíos.
- Investiga algún algoritmo de ordenación que permita optimizar el tiempo de ejecución (minimizar el número de comparaciones) y desarrolla el script basándote en él.
- Recomendable: utilización de mensajes por consola (console.log, info, etc)

## ■ EJERCICIO 2.

- En este ejercicio se pretende practicar el uso de **switch...case** y bucles, para realizar cálculos aritméticos.
- Dado un número y una operación aritmética introducida por teclado (a través de prompt) se quiere hacer la operación introducida sucesivamente sobre dicho número-1, hasta llegar a 0 (0 no incluido).
- Valores válidos de la operación: "SUMA", "RESTA", "MULTIPLICACIÓN", "DIVISIÓN".
- Ejemplo:
  - Número introducido = 9
  - Operación = SUMA
  - Deberá realizar  $9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$   
(sigue)



## Prácticas 2.

---

### **EJERCICIO 3.**

Cálculo de la letra del DNI.

El proceso sería dividir la parte numérica entre 23 y obtener su resto.

Dicho resto tendríamos que identificarlo con la posición de la siguiente lista de letras (empezando en 0). En obtener la letra.

`['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];`

Nota: El objetivo de este ejercicio es utilizar switch...case con el resto de la división y obtener los distintos casos de letras.

No deben usarse métodos del objeto array ni elementos arrays. (Se estudiarán más adelante)



## Bibliografía.

---

- Desarrollo Web en Entorno Cliente. Editorial Rama. Juan Manuel Vara, otros...
- Desarrollo web en entorno cliente: Javascript Avanzado. Juan Felix Mateos.
- Repositorios Aulas Virtuales del Ministerio de Educación y Formación Profesional. Curso a distancia: Desarrollo Web en Entorno Cliente.
- Tutorial de Javascript:  
<https://www.w3schools.com/js/default.asp>