Material Imprimible

Curso de JavaScript

Módulo 4

Funciones utilizadas como valores variables

Las funciones se pueden usar de la misma manera que las variables, en todos los tipos de fórmulas, tareas y cálculos.

En lugar de usar una variable para almacenar el valor de retorno de una función:

```
<! DOCTYPE html>
<html>
<body>
<h2> Funciones de JavaScript </h2>
<p id = "demo"> </p>
<script>
document.getElementById ("demo"). innerHTML =
"La temperatura es" + toCelsius (77) + "Celsius";

function toCelsius (fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
</script>

</body>

</html>
```

Variables locales

Las variables declaradas dentro de una función JavaScript, se convierten en **LOCAL** a la función.

Solo se puede acceder a las variables locales desde la función.

```
<! DOCTYPE html>
<html>
<body>

<h2> Funciones de JavaScript </h2>

<p> Fuera de myFunction () carName no está definido. </p>

<p id = "demo1"> </p>

<p id = "demo2"> </p>

<script>
myFunction ();

function myFunction () {
  var carName = "Volvo";
  document.getElementById ("demo1"). innerHTML =
  typeof carName + "" + carName;
}

document.getElementById ("demo2"). innerHTML =
typeof carName;
</script>

</body>
</html>
```

Dado que las variables locales solo se reconocen dentro de sus funciones, las variables con el mismo nombre se pueden utilizar en diferentes funciones.

Las variables locales se crean cuando se inicia una función y se eliminan cuando se completa la función. Ejercicio

```
function myFunction() {
 alert("Hello World!");
}
;
```

Objetos, propiedades y métodos de la vida real

En la vida real, un coche es un **objeto.**

Un coche tiene **propiedades** como el peso y el color, y **métodos** como el inicio y la parada:

Objeto	Propiedades	Métodos
	car.name - Fiat car.model á 500 car.peso á 850kg car.color á blanco	car.start() car.drive() car.brake() car.stop()

Todos los coches tienen las mismas **propiedades,** pero los **valores** de propiedad difieren de un coche a uno.

Todos los coches tienen los **mismos métodos,** pero los métodos se realizan **en diferentes momentos.**

Objetos JavaScript

Ya ha aprendido que las variables de JavaScript son contenedores para los valores de datos.

Este código asigna un **valor simple** (Fiat) a una **variable** denominada car:

```
<! DOCTYPE html>
<html>
<body>

<h2> Variables de JavaScript </h2>

<p id = "demo"> </p>

<script>
// Crea y muestra una variable:
var car = "Fiat";
document.getElementById ("demo"). innerHTML = car;
</script>

</body>
</html>
```

Los objetos también son variables. Pero los objetos pueden contener muchos valores.

Este código asigna **muchos valores** (Fiat, 500, blanco) a una **variable** denominada car:

```
<! DOCTYPE html>
<html>
<body>

<h2> Objetos JavaScript </h2>

<p id = "demo"> </p>

<script>
// Crear un objeto:
var car = {type: "Fiat", modelo: "500", color: "blanco"};
```

```
// Muestra algunos datos del objeto:
document.getElementById ("demo"). innerHTML = "El tipo de auto es" + car.type;
</script>

</body>
</html>
```

Los valores se escriben como pares **name:value** (nombre y valor separados por dos puntos).

Los objetos JavaScript son contenedores para **valores con nombre** denominados propiedades o métodos.

Definición de objeto

Defina (y cree) un objeto JavaScript con un literal de objeto:

```
<! DOCTYPE html>
<html>
<body>

<h2> Objetos JavaScript </h2>

<p id = "demo"> </p>

<script>
// Crear un objeto:
var person = {firstName: "John", lastName: "Doe", edad: 50, eyeColor: "blue"};

// Muestra algunos datos del objeto:
document.getElementById ("demo"). innerHTML =
person.firstName + "tiene" + person.edad + "años"
</script>
</body>
</html>
```

Los espacios y los saltos de línea no son importantes. Una definición de objeto puede abarcar varias líneas:

```
<! DOCTYPE html>
<html>
<body>

<h2> Objetos JavaScript </h2>

<p id = "demo"> </p>

<script>
// Crear un objeto:
var person = {
  firstName: "John",
  apellido: "Doe",
  edad: 50,
  eyeColor: "azul"
};

// Muestra algunos datos del objeto:
document.getElementById ("demo"). innerHTML =
person.firstName + " tiene " + person.edad + " años";
</script>

</body>
</html>
```

Acceso a las propiedades del objeto

Puede acceder a las propiedades del objeto de dos maneras:

objectName.propertyName

O

objectName["propertyName"]

```
<! DOCTYPE html>
<html>
<body>

<h2> Objetos JavaScript </h2>

<p> Hay dos formas diferentes de acceder a una propiedad de objeto. </p>

<p> Puede usar person.property o person ["property"]. </p>

<p id = "demo"> </p>

<script>
// Crear un objeto:
var persona = {
  firstName: "John",
  apellido: "Doe",
  id: 5566
};
// Muestra algunos datos del objeto:
document.getElementById ("demo"). innerHTML =
persona.firstName + " " + persona ["apellido"];
</script>

</body>
</html>
<! DOCTYPE html>
<html>
<body>
```

```
<h2> Objetos JavaScript </h2>

<p> Hay dos formas diferentes de acceder a una propiedad de objeto. </p>

<p> Puede usar person.property o person ["property"]. </p>

<p id = "demo"> </p>

<script>
// Crear un objeto:
var persona = {
  firstName: "John",
  apellido: "Doe",
  id: 5566
};
// Muestra algunos datos del objeto:
document.getElementById ("demo"). innerHTML =
persona ["firstName"] + "" + persona ["apellido"];
</script>

</body>
</html>
```

Métodos de objeto

Los objetos también pueden tener **métodos.**

Los métodos son **acciones** que se pueden realizar en objetos.

Los métodos se almacenan en propiedades como definiciones de **función.**

Propiedad	Valor de la propiedad
Apellido	John
Apellido	Doe
Edad	50
eyeColor	Azul
Fullname	function() devuelve this.firstName + " " + this.lastName;?

Un método es una función almacenada como una propiedad.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
            return this.firstName + " " + this.lastName;
  }
};
```

La palabra clave this

En una definición de función, this hace referencia al "propietario" de la función.

En el ejemplo anterior, thisfullName es el **objeto person** que "posee" la función.

En otras palabras, this significa la propiedad firstName de **este objeto.**

Acceso a métodos de objeto

Puede acceder a un método de objeto con la sintaxis siguiente:

objectName.methodName()

```
<! DOCTYPE html>
<html>
```

```
<body>

<h2> Objetos JavaScript </h2>

<p> Un método de objeto es una definición de función, almacenada como un valor de
propiedad. </p>

<p id = "demo"> </p>

<script>
// Crear un objeto:
var person = {
  firstName: "John",
  apellido: "Doe",
  id: 5566,
  fullName: function () {
          return this.firstName + " " + this.apellido;
  }
};
// Mostrar datos del objeto:
document.getElementById ("demo"). innerHTML = person.fullName ();
</script>

</body>
</html>
```

¡No declare cadenas(String), números y booleanos como objetos!

Cuando se declara una variable JavaScript con la palabra clave " new ", la variable se crea como un objeto:

```
var x = new String();       // Declares x as a String object
var y = new Number();       // Declares y as a Number object
```

```
var z = new Boolean();      // Declares z as a Boolean object
```

Evitar `String`, `Number` , `Boolean` y objetos. Complican el código y ralentizan la velocidad de ejecución.

```
var person = {
 firstName: "John",
 lastName: "Doe"
};

alert();
```

Eventos JavaScript

Los eventos HTML son **"cosas"** que suceden con los elementos HTML.

Cuando JavaScript se utiliza en páginas HTML, JavaScript puede **"reaccionar"** en estos eventos.

Eventos HTML

Un evento HTML puede ser algo que hace el navegador o algo que hace un usuario.

Estos son algunos ejemplos de eventos HTML:

- Una página web HTML ha terminado de cargarse
- Se ha cambiado un campo de entrada HTML
- Se hizo clic en un botón HTML

A menudo, cuando ocurren eventos, es posible que desee hacer algo.

JavaScript le permite ejecutar código cuando se detectan eventos.

HTML permite agregar atributos de controlador de eventos, **con código JavaScript,**a los elementos HTML.

Con comillas simples:

```
<element event='some JavaScript'>
```

Con comillas dobles:

```
<element event="some JavaScript">
```

En el ejemplo siguiente, se agrega un atributo (con código) a un elemento:`onclick<button>`

```
<! DOCTYPE html>
<html>
```

```
<body>

<button onclick = "document.getElementById ('demo'). innerHTML = Date ()"> ¿Es la hora?
</button>

<p id = "demo"> </p>

</body>
</html>
```

En el ejemplo anterior, el código JavaScript cambia el contenido del elemento con id-"demo".

En el siguiente ejemplo, el código cambia el contenido de su propio elemento (utilizando):

this.innerHTML

```
<! DOCTYPE html>
<html>
<body>

<button onclick = "this.innerHTML = Date ()"> ¿La hora es? </button>

</body>
</html>
```

El código JavaScript suele ser de varias líneas de largo. Es más común ver atributos de evento que llaman a funciones:

```
<! DOCTYPE html>
<html>
<body>

<p> Haga clic en el botón para mostrar la fecha. </p>

<button onclick = "displayDate ()"> ¿Es la hora? </button>
```

```
<script>
function displayDate () {
  document.getElementById ("demo"). innerHTML = Date ();
}
</script>

<p id = "demo"> </p>

</body>
</html>
```

¿Qué puede hacer JavaScript?

Los controladores de eventos se pueden usar para controlar y comprobar la entrada del usuario, las acciones del usuario y las acciones del explorador:

- Cosas que se deben hacer cada vez que se carga una página
- Cosas que se deben hacer cuando se cierra la página
- Acción que se debe realizar cuando un usuario hace clic en un botón
- Contenido que debe verificarse cuando un usuario introduce datos
- Y más ...

Se pueden utilizar muchos métodos diferentes para permitir que JavaScript funcione con eventos:

- Los atributos de eventos HTML pueden ejecutar código JavaScript directamente
- Los atributos de eventos HTML pueden llamar a funciones JavaScript
- Puede asignar sus propias funciones de controlador de eventos a elementos HTML
- Puede evitar que se envíen o manipulen eventos
- Y más ...

Ejercicio:

El elemento <button> debe hacer algo cuando alguien hace clic en él. ¡Trata de arreglarlo!

```
<button  ="alert('Hello')">Click me.</button>
```

Onclick

Bucles y condiciones en JavaScript

IF es una estructura de control utilizada para tomar decisiones. Es un condicional que sirve para realizar unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo.

La sintaxis de la estructura IF es la siguiente.

Nota: Todas las estructuras de control se escriben en minúsculas en Javascript. Aunque algunas veces para destacar el nombre de la estructura la podamos escribir en el texto del manual con letras mayúsculas, en el código de nuestros scripts siempre tenemos que ponerlo en minúsculas. En caso contrario recibiremos un mensaje de error.

```
if (expresión) {
  //acciones a realizar en caso positivo
  //...
}
```

Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia devuelva resultados negativos.

```
if (expresión) {
  //acciones a realizar en caso positivo
  //...
} else {
  //acciones a realizar en caso negativo
  //...
}
```

Fijémonos en varias cosas. Para empezar, vemos como con unas llaves engloban las acciones que queremos realizar en caso de que se cumplan o no las expresiones. Estas llaves han de colocarse siempre, excepto en el caso de que sólo haya una instrucción como acciones a realizar, que son opcionales.

Nota: Aunque las llaves para englobar las sentencias a ejecutar tanto en el caso positivo como negativo sean opcionales cuando queremos ejecutar una única sentencia, la recomendación es colocarlas siempre, porque obtendremos así un código fuente más claro. Por ejemplo:

```
if (llueve)
  alert("Cae agua");
```

Sería exactamente igual que este código:

```
if (llueve){
  alert("Cae agua");
}
```

O incluso, igual a este otro:

```
if (llueve) alert("Cae agua");
```

Sin embargo, cuando utilizamos las llaves, el código queda bastante más claro, porque se puede apreciar en un rápido vistazo qué instrucciones están dependiendo del caso positivo del if. Esto es un detalle que ahora quizás no tenga mucha importancia, pero que se agradecerá cuando el programa sea más complejo o cuando varios programadores se encarguen de tocar un mismo código.

Veamos algún ejemplo de condicionales IF.

```
if (dia == "lunes")
  document.write ("Que tengas un feliz comienzo de semana")
```

Si es lunes nos deseará una feliz semana. No hará nada en caso contrario. Como en este ejemplo sólo indicamos una instrucción para el caso positivo, no hará falta utilizar las llaves (aunque sí sería recomendable haberlas puesto). Fíjate también en el operador condicional que consta de dos signos igual.

Vamos a ver ahora otro ejemplo, un poco más largo.

```
if (credito >= precio) {
```

```
  document.write("has comprado el artículo " + nuevoArtículo) //enseño compra
  carrito += nuevoArticulo //introduzco el artículo en el carrito de la compra
  credito -= precio //disminuyo el crédito según el precio del artículo
} else {
  document.write("se te ha acabado el crédito") //informo que te falta dinero
  window.location = "carritodelacompra.html" //voy a la página del carrito
}
```

Este ejemplo es un poco más complejo, y también un poco ficticio. Lo que hago es comprobar si tengo crédito para realizar una supuesta compra. Para ello miro si el crédito es mayor o igual que el precio del artículo, si es así informo de la compra, introduzco el artículo en el carrito y resto el precio al crédito acumulado. Si el precio del artículo es superior al dinero disponible informo de la situación y mando al navegador a la página donde se muestra su carrito de la compra.

Expresiones condicionales

La expresión a evaluar se coloca siempre entre paréntesis y está compuesta por variables que se combinan entre si mediante operadores condicionales. Recordamos que los operadores condicionales relacionaban dos variables y devolvían siempre un resultado boleano. Por ejemplo, un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

```
if (edad > 18)
  document.write("puedes ver esta página para adultos")
```

En este ejemplo utilizamos en operador condicional "es mayor" (>). En este caso, devuelve true si la variable edad es mayor que 18, con lo que se ejecutaría la línea siguiente que nos informa de que se puede ver el contenido para adultos.

Las expresiones condicionales se pueden combinar con las expresiones lógicas para crear expresiones más complejas. Recordamos que las expresiones lógicas son las que tienen como operandos a los boleanos y que devuelvan otro valor boleano. Son los operadores negación lógica, Y lógico y O lógico.

```
if (bateria < 0.5 && redElectrica == 0)
  document.write("tu ordenador portátil se va a apagar en segundos")
```

Lo que hacemos es comprobar si la batería de nuestro supuesto ordenador es menor que 0.5 (está casi acabada) y también comprobamos si el ordenador no tiene red eléctrica (está desenchufado). Luego, el operador lógico los relaciona con un Y, de modo que si está casi sin batería Y sin red eléctrica, informo que el ordenador se va a apagar.

Sentencias IF anidadas

Para hacer estructuras condicionales más complejas podemos anidar sentencias IF, es decir, colocar estructuras IF dentro de otras estructuras IF. Con un solo IF podemos evaluar y realizar una acción u otra según dos posibilidades, pero si tenemos más posibilidades que evaluar debemos anidar IFs para crear el flujo de código necesario para decidir correctamente.

Por ejemplo, si deseo comprobar si un número es mayor menor o igual que otro, tengo que evaluar tres posibilidades distintas. Primero puedo comprobar si los dos números son iguales, si lo son, ya he resuelto el problema, pero si no son iguales todavía tendré que ver cuál de los dos es mayor. Veamos este ejemplo en código Javascript.

```
var numero1=23
var numero2=63
if (numero1 == numero2){
  document.write("Los dos números son iguales")
}else{
  if (numero1 > numero2) {
    document.write("El primer número es mayor que el segundo")
  }else{
    document.write("El primer número es menor que el segundo")
  }
}
```

El flujo del programa es como comentábamos antes, primero se evalúa si los dos números son iguales. En caso positivo se muestra un mensaje informando de ello. En caso contrario ya

sabemos que son distintos, pero aun debemos averiguar cuál de los dos es mayor. Para eso se hace otra comparación para saber si el primero es mayor que el segundo. Si esta comparación da resultados positivos mostramos un mensaje diciendo que el primero es mayor que el segundo, en caso contrario indicaremos que el primero es menor que el segundo.

Volvemos a remarcar que las llaves son en este caso opcionales, pues sólo se ejecuta una sentencia para cada caso. Además, los saltos de línea y las sangrías también opcionales en todo caso y nos sirven sólo para ver el código de una manera más ordenada. Mantener el código bien estructurado y escrito de una manera comprensible es muy importante, ya que nos hará la vida más agradable a la hora de programar y más adelante cuando tengamos que revisar los programas.

Operador IF

Hay un operador que no hemos visto todavía y es una forma más esquemática de realizar algunos IF sencillos. Proviene del lenguaje C, donde se escriben muy pocas líneas de código y donde cuanto menos escribamos más elegantes seremos. Este operador es un claro ejemplo de ahorro de líneas y caracteres al escribir los scripts. Lo veremos rápidamente, pues la única razón por la que lo incluyo es para que sepas que existe y si lo encuentras en alguna ocasión por ahí sepas identificarlo y cómo funciona.

Un ejemplo de uso del operador IF se puede ver a continuación.

Variable = (condición) ? valor1 : valor2;

Este ejemplo no sólo realiza una comparación de valores, además asigna un valor a una variable. Lo que hace es evaluar la condición (colocada entre paréntesis) y si es positiva asigna el valor1 a la variable y en caso contrario le asigna el valor2. Veamos un ejemplo:

momento = (hora_actual < 12) ? "Antes del mediodía" : "Después del mediodía"

Este ejemplo mira si la hora actual es menor que 12. Si es así, es que ahora es antes del mediodía, así que asigna "Antes del mediodía" a la variable momento. Si la hora es mayor o igual a 12 es

que ya es después de mediodía, con lo que se asigna el texto "Después del mediodía" a la variable momento

Bucle FOR

El bucle FOR se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro la cantidad de veces. La sintaxis del bucle for se muestra a continuación.

```
for (inicialización; condición; actualización) {
        //sentencias a ejecutar en cada iteración
}
```

El bucle FOR tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.
La segunda parte es la condición, que se evaluará cada vez que comience una iteración del bucle. Contiene una expresión para decidir cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.
Por último, tenemos la actualización, que sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.
Después del FOR se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Un ejemplo de utilización de este bucle lo podemos ver a continuación, donde se imprimirán los números del 0 al 10.

```
var i ;
for (i=0;i<=10;i++) {
        document.write(i);
        document.write("<br>") ;
```

}

En este caso se inicializa la variable i a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Como se puede comprobar, este bucle es muy potente, ya que en una sola línea podemos indicar muchas cosas distintas y muy variadas, lo que permite una rápida configuración del bucle y una versatilidad enorme.

Por ejemplo, si queremos escribir los número del 1 al 1.000 de dos en dos se escribirá el siguiente bucle.

```
for (i=1;i<=1000;i+=2)
        document.write(i)
```

Si nos fijamos, en cada iteración actualizamos el valor de i incrementándolo en 2 unidades.

Nota: Otro detalle, no utilizamos las llaves englobando las instrucciones del bucle FOR porque sólo tiene una sentencia y en este caso no es obligado, tal como pasaba con las instrucciones del IF.

Si queremos contar descendentemente del 343 al 10 utilizaríamos este bucle.

```
for (i=343;i>=10;i--)
        document.write(i)
```

En este caso decrementamos en una unidad la variable i en cada iteración, comenzando en el valor 343 y siempre que la variable tenga un valor mayor o igual que 10.

Ejercicio de ejemplo del bucle for

Vamos a hacer una pausa para asimilar el bucle for con un ejercicio que no encierra ninguna dificultad si hemos entendido el funcionamiento del bucle.

Se trata de hacer un bucle que escriba en una página web los encabezamientos desde <H1> hasta <H6> con un texto que ponga "Encabezado de nivel x".

Lo que deseamos escribir en una página web mediante JavaScript es lo siguiente:

<H1>Encabezado de nivel 1</H1>
<H2>Encabezado de nivel 2</H2>
<H3>Encabezado de nivel 3</H3>
<H4>Encabezado de nivel 4</H4>
<H5>Encabezado de nivel 5</H5>
<H6>Encabezado de nivel 6</H6>

Para ello tenemos que hacer un bucle que empiece en 1 y termine en 6 y en cada iteración escribiremos el encabezado que toca.

```
<script>
for (i=1;i<=6;i++) {
        document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">")
}
</script>
```

Bucle WHILE

Estos bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Se más sencillo de comprender que el bucle FOR, pues no incorpora en la misma línea la inicialización de las variables su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración.

```
while (condición){
        //sentencias a ejecutar
}
```

Un ejemplo de código donde se utiliza este bucle se puede ver a continuación.

```
var color = ""
while (color != "rojo"){
        color = prompt("dame un color (escribe rojo para salir)","")
```

```
}
```

Este es un ejemplo de lo más sencillo que se puede hacer con un bucle while. Lo que hace es pedir que el usuario introduzca un color y lo hace repetidas veces, mientras que el color introducido no sea rojo. Para ejecutar un bucle como este primero tenemos que inicializar la variable que vamos utilizar en la condición de iteración del bucle. Con la variable inicializada podemos escribir el bucle, que comprobará para ejecutarse que la variable color sea distinto de "rojo". En cada iteración del bucle se pide un nuevo color al usuario para actualizar la variable color y se termina la iteración, con lo que retornamos al principio del bucle, donde tenemos que volver a evaluar si lo que hay en la variable color es "rojo" y así sucesivamente mientras que no se haya introducido como color el texto "rojo".

Bucle DO...WHILE

El bucle do...while es la última de las estructuras para implementar repeticiones de las que dispone en Javascript y es una variación del bucle while visto anteriormente. Se utiliza generalmente cuando no sabemos cuantas veces se habrá de ejecutar el bucle, igual que el bucle WHILE, con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

La sintaxis es la siguiente:

```
do {
        //sentencias del bucle
} while (condición)
```

El bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución.

Veamos el ejemplo que escribimos para un bucle WHILE en este otro tipo de bucle.

```
var color;
do {
        color = prompt("dame un color (escribe rojo para salir)","") ;
```

```
} while (color != "rojo")
```

Este ejemplo funciona exactamente igual que el anterior, excepto que no tuvimos que inicializar la variable color antes de introducirnos en el bucle. Pide un color mientras que el color introducido es distinto que "rojo".

Ejemplo de uso de los bucles WHILE

Vamos a ver a continuación un ejemplo más práctico sobre cómo trabajar con un bucle WHILE. Como resulta muy difícil hacer ejemplos prácticos con lo poco que sabemos sobre Javascript, vamos a adelantar una instrucción que aun no conocemos.

En este ejemplo vamos a declarar una variable e inicializarla a 0. Luego iremos sumando a esa variable un número aleatorio del 1 al 100 hasta que sumemos 1.000 o más, imprimiendo el valor de la variable suma después de cada operación. Será necesario utilizar el bucle WHILE porque no sabemos exactamente el número de iteraciones que tendremos que realizar (dependerá de los valores aleatorios que se vayan obteniendo).

```
var suma = 0
while (suma < 1000){
        suma += parseInt(Math.random() * 100)
        document.write (suma + "<br>")
}
```

Suponemos que por lo que respecta al bucle WHILE no habrá problemas, pero donde si que puede haberlos es en la sentencia utilizada para tomar un número aleatorio. Sin embargo, no es necesario explicar aquí la sentencia porque lo tenemos planeado hacer más adelante.

Fuentes:

https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/JavaScript_basics