U1. Navegadores Web. Lenguajes Cliente. Sintaxis Javascript. Parte II

Desarrollo Web en Entorno Cliente

Índice

- Funciones
  - Concepto función
  - Creación de funciones
  - Invocación de funciones
  - Valores por defecto
  - Asignar función a variable
  - Ámbito de las variables
  - Ejemplos funciones predefinidas
- Arrays
  - Utilidad de los arrays
  - Sintaxis
  - Propiedad length
  - Uso de arrays

Funciones

- Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta.
  - Permiten la reutilización.
  - Facilitan la modificación de código.
  - Ayudan a la estructuración del código y la comprensión del código.

Creación de funciones

- Definición de funciones – Sintaxis:

```
function nombre_función ([argumentos]){
  grupo_de_instrucciones;
  [return valor;]
}
```

Ejemplo:

```
function saluda() {
        alert("Hola!");
}
```

Para definir el nombre de la función debemos tener en cuenta que:

- Deben usarse sólo letras, números o el carácter de subrayado.
- Debe ser único en el código JavaScript de la página web.
- No pueden empezar por un número.
- No puede ser palabras reservadas del lenguaje

Creación de funciones

- Ejemplo – Función que calcula el importe de un producto después de haberle aplicado el IVA:

```
function aplicarIVA(valorProducto, IVA){
  var productoConIVA = valorProducto * IVA;
  alert("El precio del producto con IVA"
          + IVA + " es: "
          + productoConIVA");
  return productoConIVA;
}
```

Invocación de funciones

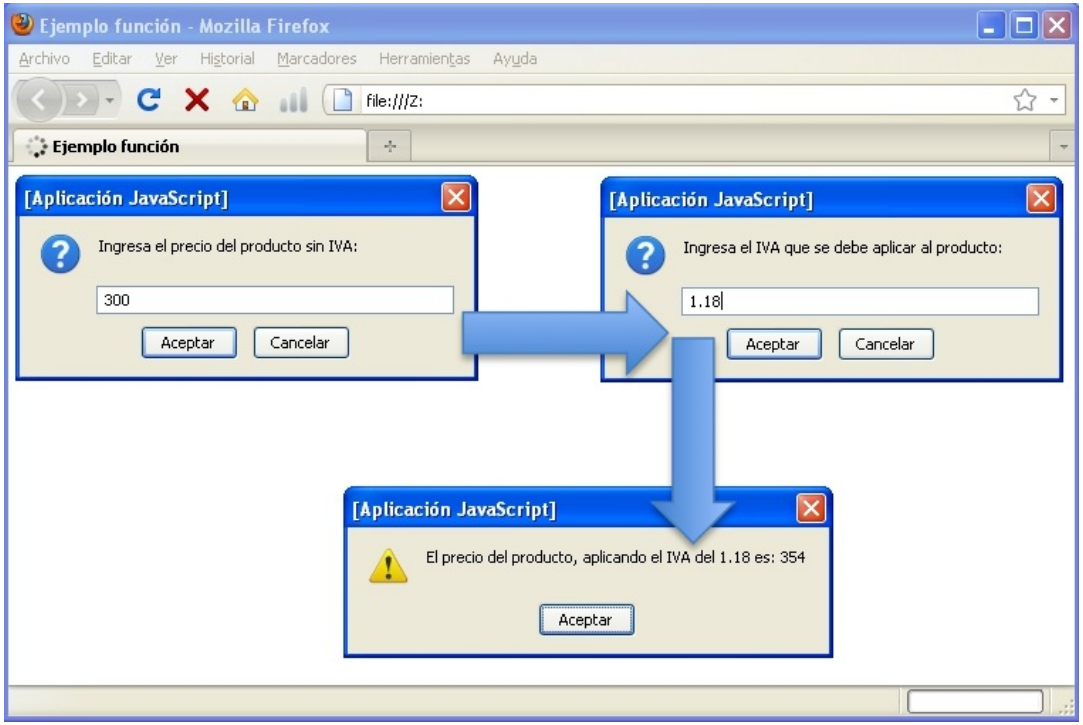
- Invocación de funciones:
  - Una vez definida la función es necesaria llamarla para que el navegador ejecute el grupo de instrucciones.
  - Se invoca usando su nombre seguido del paréntesis.
  - Si tiene argumentos, se deben especificar en el mismo orden en el que se han definido en la función.

Ejemplo: saluda();

Ejemplo: precio = aplicarIVA(x,y);

Invocación de funciones

- Ej: `precio = aplicarIVA(300, 1.18);`



Valores por defecto

- El operador boolean || (or) puede ser usado la para simular valores por defecto en una función. Por ejemplo:

```
function sayHello(name) {
// Si name es undefined o vacío, asignará "Anonymous"
  var sayName = name || "Anonymous";
  console.log("Hello " + sayName);
}
sayHello("Peter"); // Imprime "Hello Peter"
sayHello(); // Imprime "Hello Anonymous"
```

Asignar funciones a variables

- JS permite asignar funciones a variables de la siguiente manera:

```
var a = function (){
  alert(“Hola”);
}
```

- Podemos invocar la función a través de la variable así:

```
a();
```

- Si hacemos var b = a; b pasa a ser también una función.

Ámbito de las variables

- Por defecto, las variables son globales. Es decir, se pueden utilizar en todas partes (dentro y fuera de cualquier función).
- Aunque empleemos la variable sólo en la función, si no ponemos var (ej: x= 5), la variable es global.
- Las funciones nos permiten definir variable locales y emplear paso de parámetros para intercambiar valores.
- Las variables locales, sólo existen allí donde han sido definidas.

Ámbito de las variables

Ámbito Global

Las variables y funciones definidas aquí están disponibles en **todas** partes.

Ámbito local (dentro de la FUNCIÓN)

Las variables y funciones definidas aquí **sólo** están disponibles **dentro de la función.**

¿Pueden existir 2 variables con el mismo nombre en distinto ámbito?

Ámbito de las variables

- Cada vez que se ejecuta una función crea **su contexto de ejecución**.
- Cada contexto de ejecución **contiene**:
  - Sus **variables locales**
  - Una referencia al contexto de ejecución externo
  - La referencia al objeto especial **this**
- El ámbito global es el contexto de ejecución más externo.
- Cuando una función solicite una variable que no esté definida en su contexto, buscará en el exterior sucesivamente hasta encontrarla o la declarará undefined.

Hoisting

- El Hoisting hace referencia a que antes de ejecutar el código, JS lo analiza y realiza dos cosas:
  - Carga la declaración de funciones en memoria (por tanto, pueden ser accesibles desde cualquier posición).
  - Mueve la declaración de variables al principio de las funciones/bloque principal.

```
function printHello() {
  console.log(hello);
  var hello = "Hello World";}

printHello(); // Esto imprimirá undefined y debería imprimir un error
```

En realidad, internamente JS ha transformado el código en este:

```
function printHello() {
  var hello = undefined;
  console.log(hello);
  hello = "Hello World"; }
printHello();
```

Funciones como parámetros de funciones

- JS aplica a las funciones el mismo tratamiento que a una variable por lo que podemos emplear una función como parámetro de otra.
- Ejemplo:

```
function saludador(persona){
        alert("Buenos días  " + persona);
}
function saludaPacos(saludador){
        var nombre = "Paco";
        saludador(nombre);
}
saludaPacos(saludador);
```

Funciones predefinidas del lenguaje

- Las siguientes son algunas de las principales funciones predefinidas de JavaScript:

Funciones Predefinidas	
`escape()`	`isNaN()`
`eval()`	`parseInt()`
`isFinite()`	`parseFloat()`

Funciones predefinidas del lenguaje

- `escape()`: recibe como argumento una cadena de caracteres y devuelve esa misma cadena sustituida con su codificación en ASCII.

```
<script type="text/javascript">
  var input = prompt("Introduce una cadena");
  var inputCodificado = escape(input);
  alert("Cadena codificada: " + inputCodificado);
</script>
```

Funciones predefinidas del lenguaje

- `eval()`: convierte una cadena que pasamos como argumento en código JavaScript ejecutable.

```
<script type="text/javascript">
  var input = prompt("Introduce una operación numérica");
  var resultado = eval(input);
  alert ("El resultado de la operación es: " + resultado);
</script>
```

Funciones predefinidas del lenguaje

- `isFinite()`: verifica si el número que pasamos como argumento es o no un número finito.

```
if(isFinite(argumento)) {
  //instrucciones si el argumento es un número finito
}else{
  //instrucciones si el argumento no es un número finito
}
```

Funciones predefinidas del lenguaje

- `isNaN()`: comprueba si el valor que pasamos como argumento es un de tipo numérico.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor numérico: ");
  if (isNaN(input)){
    alert("El dato ingresado no es numérico.");
  }else{
    alert("El dato ingresado es numérico.");
  }
</script>
```

Funciones predefinidas del lenguaje

- `parseInt()`: convierte la cadena que pasamos como argumento en un valor numérico de tipo entero.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor: ");
  var inputParsed = parseInt(input);
  alert("parseInt("+input+"): "+inputParsed);
</script>
```

Funciones predefinidas del lenguaje

- `parseFloat()`: convierte la cadena que pasamos como argumento en un valor numérico de tipo flotante.

```
<script type="text/javascript">
  var input = prompt("Introduce un valor: ");
  var inputParsed = parseFloat(input);
  alert("parseFloat("+input+"): " + inputParsed);
</script>
```

Arrays

- La mayor parte de las aplicaciones web gestionan un número elevado de datos.
- Por ejemplos si se quisiera definir el nombre de 180 productos alimenticios:

```
var producto1 = “Pan”;
var producto2 = “Agua”;
var producto3 = “Lentejas”;
var producto4 = “Naranjas”;
var producto5 = “Cereales”;
...
var producto180 = “Salsa agridulce”;
```

Utilidad de los Arrays

- Si posteriormente se quisiera mostrar el nombre de estos productos:

```
document.write(producto1);
document.write(producto2);
document.write(producto3);
document.write(producto4);
document.write(producto5);
...
document.write(producto180);
```

Utilidad de los Arrays

- El anterior ejemplo es correcto, pero sería una tarea compleja, repetitiva y propensa a errores.
- Para gestionar este tipo de escenarios se pueden utilizar los arrays.
- Un array es un conjunto ordenado de valores relacionados.

```
nombreArray[índice] = valorElemento;
```

  - Cada uno de estos valores es un elemento del array.
  - Cada elemento tiene un índice que indica su posición numérica en el array.
  - El primer índice del array es el 0.

Sintaxis Array

- La declaración de un array se hace de la misma forma que se declara cualquier variable:

  var nombreDelArray;

- El array adquiere condición de tal cuando
  a. la variable se inicializa con forma de array con un contenido inicial

     var nombreDelArray = [‘a’, ’b‘];

  b. la variable se inicializa como array vacío

     var nombreDelArray = [];

  y posteriormente añado elementos

     nombreDelArray[1] = ‘b’;

Veremos más adelante la relación entre los arrays y los objetos.

Sintaxis Array

- Ejemplo:
  - Declaro:

    var pais = ['Mexico', 'España', 'Argentina', 'Chile', 'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];
  - Accedo a los valores:

    pais[0]

    pais[1]

    pais[2]

    pais[3]

    pais[4]

    pais[5]

    pais[6]

    pais[7]

Sintaxis arrays

- Mediante el uso de un bucle se pueden escribir instrucciones mucho más limpias y eficientes:

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

Sintaxis Array

- Normalmente los arrays contendrán elementos de un tipo, por ejemplo valores numéricos, pero en ocasiones nos interesará que contengan elementos de distintos tipos.
- Ejemplo

var datos = ['Frío', 33, false, 'nube', -11.22, true, 3.33, 'variado'];

  - El elemento de índice cero es de tipo texto
  - El índice 1 es un valor numérico
  - El elemento de índice tres es un valor booleano, etc.

Propiedad length

- Los arrays tiene la propiedad **length** que define el número de elementos que contienen

- Podemos emplearla con la sintaxis siguiente:

  nombreArray.length

Ejemplo:

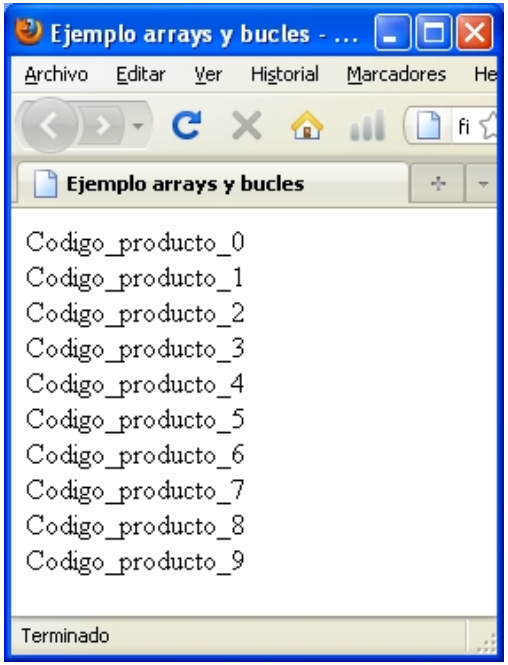
var pais = ['Mexico', 'España', 'Argentina', 'Chile', 'Colombia', 'Venezuela', 'Perú', 'Costa Rica'];

```
> pais.length
< 8
```

Uso de Arrays

Habitualmente, emplearemos bucles para recorrer los elementos de un Array de la siguiente manera:

```
for (var i=0; i<codigos_productos.length; i++){
  document.write(codigos_productos[i] + "<br>");
}
```



Uso de Arrays

- Otra versión del for, es el bucle for..in.
- Este bucle podemos iterar los índices de un array o las propiedades de un objeto (similar al bucle foreach de otros lenguajes, pero recorriendo índices en lugar de valores).

```
var ar = new Array(4, 21, 33, 24, 8);
for (var index in ar) { // Imprime 4 21 33 24 8
  console.log(ar[index]);
}
```