Programación Web
Tema 3.2 Java Script – Estructura del Lenguaje

Miguel Ángel Manso
Emerson Castañeda
Ramón Alcarria
ETSI en Topografía, Geodesia y Cartografía - UPM

Contenido

- Variables
- Operadores
- Estructuras de control de flujo
- Funciones y propiedades básicas
- Funciones
- Sentencias *break* y *continue*
- Otras estructuras de control
- Eventos

Variables (I)

- Las variables en JavaScript se crean mediante la palabra reservada **var**
- La palabra reservada **var** solamente se debe indicar al definir por primera vez la variable, lo que se denomina **declarar** una variable
- El nombre de una variable también se conoce como **identificador** y debe cumplir las siguientes normas:
  - Sólo puede estar formado por letras, números y los símbolos $ (dólar) y _ (guión bajo)
  - El primer carácter no puede ser un número

Variables (II)

- Si cuando se declara una variable se le asigna también un valor, se dice que la variable ha sido **inicializada**
- En **JavaScript** *no es obligatorio* inicializar las variables, se pueden declarar por una parte y asignarles un valor posteriormente
- Una de las características destacables de **JavaScript** es que tampoco es necesario declarar las variables. *Se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada* ***var***

Variables (III)

- Aunque todas las variables de JavaScript se crean de la misma forma, la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)
- Numéricas: Se utilizan para almacenar valores numéricos enteros (*integer*) o decimales (*float*)
  - el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter . (punto) en vez de , (coma) para separar la parte entera y la parte decimal

```
// variable tipo entero
var iva = 16;
// variable tipo decimal
var total = 234.65;
```


Variables (IV)

- Cadenas de texto: Se utilizan para almacenar caracteres, palabras y/o frases de texto
- Para asignar el valor se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
```

```
/* El contenido de texto1 tiene comillas simples, por lo que
se encierra con comillas dobles */
 var texto1 = "Una frase con 'comillas simples' dentro";
/* El contenido de texto2 tiene comillas dobles, por lo que
se encierra con comillas simples */
 var texto2 = 'Una frase con "comillas dobles" dentro';
```

Variables (V)

- Booleanos: También se conocen con el nombre de variables de tipo lógico. Almacena un tipo especial de valor que solamente puede tomar dos valores: *true* (verdadero) o *false* (falso)
- No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto

```
var clienteRegistrado = false;
var ivaIncluido = true;
```


Variables (VI)

- Arrays: es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente
- También se les llama vectores, matrices o arreglos

```
var nombre_array = [valor1, valor2, ..., valorN];

var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado","Domingo"];
var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"
var otroDia = dias[5]; // otroDia = "Sábado"
```

Operadores (I)

- Los operadores permiten:
  - Manipular el valor de las variables
  - Realizar operaciones matemáticas con sus valores
  - Comparar diferentes variables
  - Realizar cálculos complejos
  - Tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones

Operadores (II)

- **Asignación:** El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es =

```
var numero1 = 3;
var numero2 = 4;
/* Error, la asignación siempre se realiza a una variable,
por lo que en la izquierda no se puede indicar un número */
5 = numero1;
// Ahora, la variable numero1 vale 5
numero1 = 5;
// Ahora, la variable numero1 vale 4
numero1 = numero2;
```

Operadores (III)

- **Incremento – Decremento:** Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable

```
var numero = 5;
++numero;
alert(numero); // numero = 6

var numero = 5;
--numero;
alert(numero); // numero = 4


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 (IV)

- **Lógicos:** Sirven para realizar aplicaciones complejas, se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones. El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano

```
var visible = true;
alert(!visible); // Muestra "false" y no "true"
```

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

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

Operadores (V)

- **Matemáticos:** permiten realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*), división (/) y modulo (%)

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

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

```
var numero1 = 10;
var numero2 = 5;
resultado = numero1 % numero2; // resultado = 0
numero1 = 9;
numero2 = 5;
resultado = numero1 % numero2; // resultado = 4
```

```
var numero1 = 5;
numero1 += 3; // numero1 = numero1 + 3 = 8
numero1 -= 1; // numero1 = numero1 - 1 = 4
numero1 *= 2; // numero1 = numero1 * 2 = 10
numero1 /= 5; // numero1 = numero1 / 5 = 1
numero1 %= 4; // numero1 = numero1 % 4 = 1
```


Operadores (VI)

- **Relacionales:** Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=)

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

Operadores (VII)

```
// El operador "=" asigna valores
var numero1 = 5;
resultado = numero1 = 3; // numero1 = 3 y resultado = 3
// El operador "==" compara variables
var numero1 = 5;
resultado = numero1 == 3; // numero1 = 5 y resultado = false
```

```
var texto1 = "hola";
var texto2 = "hola";
var texto3 = "adios";
resultado = texto1 == texto3; // resultado = false
resultado = texto1 != texto2; // resultado = false
resultado = texto3 >= texto2; // resultado = false
```


Estructuras de Control (I)

- ***if*:** Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {
...
}
```

- Si la **condición** se cumple se ejecutan todas las instrucciones que se encuentran dentro de **{...}**. Si la **condición** no se cumple no se ejecuta ninguna instrucción contenida en **{...}** y el programa continúa ejecutando el resto de instrucciones del script

Estructuras de Control (II)

- ***IF … ELSE*:** Es una variante de la estructura ***IF***. Para cuando las condiciones son del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". Su definición formal es la siguiente:

```
if(condicion) {
...
}
else {
...
}
```

- Si la condición se cumple (si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del *if*(). Si la condición no se cumple (si su valor es false) se ejecutan todas las instrucciones contenidas en *else* { }

Estructuras de Control (III)

- ***FOR*:** La estructura ***for*** permite realizar repeticiones (también llamadas bucles) de una forma muy sencilla. Su definición formal es:

```
for(inicializacion; condicion; actualizacion) {
...
}
```

- El funcionamiento de un bucle *for* es el siguiente:
- "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del ***for***. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado","Domingo"];
for(var i=0; i<7; i++) {
    alert(dias[i]);
}
```

Estructuras de Control (IV)

- ***FOR … IN*:** Su definición implica el uso de objetos, un ejemplo es usar la estructura ***for...in*** con arrays y su definición formal es:

```
for(indice in array) {
...
}
```

- Si se quieren recorrer todos los elementos que forman un *array*, la estructura *for...in* es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado","Domingo"];
for(i in dias) {
    alert(dias[i]);
}
```


Funciones y Propiedades Básicas (I)

- **JavaScript** incorpora una serie de herramientas y utilidades, conocidas como **funciones** y **propiedades**, que sirven para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente
- Categorías:
  - Funciones útiles para cadenas de texto
  - Funciones útiles para arrays
  - Funciones útiles para números

Funciones y Propiedades Básicas (II)

- Funciones útiles para cadenas de texto:
  - *Length()*, calcula la longitud de una cadena de texto

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length; // numeroLetras = 10
```

  - +, se emplea para concatenar varias cadenas de texto ó la función *concat*()

```
var mensaje1 = "Hola";
var mensaje2 = " Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "Hola Mundo"

var mensaje1 = "Hola";
var mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 = "Hola Mundo"

var variable1 = "Hola ";
var variable2 = 3;
var mensaje = variable1 + variable2; // mensaje = "Hola 3"

var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + " " + mensaje2; // mensaje = "Hola Mundo"
```


Funciones y Propiedades Básicas (II)

- *toUpperCase*(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
```

- *toLowerCase*(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas

```
var mensaje1 = "HolA";
var mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

- *charAt*(posicion), obtiene el carácter que se encuentra en la posición indicada

```
var mensaje = "Hola";
var letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2); // letra = l
```

- *indexOf*(caracter), calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto.

```
var mensaje = "Hola";
var posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b'); // posicion = -1
```

Funciones y Propiedades Básicas (III)

- *lastIndexOf*() comienza su búsqueda desde el final de la cadena hacia el principio

```
var mensaje = "Hola";
var posicion = mensaje.lastIndexOf('a'); // posicion = 3
posicion = mensaje.lastIndexOf('b'); // posicion = -1
```

- *substring*(inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5); // porcion = "Mundo"
porcion = mensaje.substring(7); // porcion = "ndo"

var mensaje = "Hola Mundo";
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(3, 4); // porcion = "a"

var mensaje = "Hola Mundo";
var porcion = mensaje.substring(5, 0); // porcion = "Hola "
porcion = mensaje.substring(0, 5); // porcion = "Hola "
```


Funciones y Propiedades Básicas (III)

- *split*(separador), convierte una cadena de texto en un *array* de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" "); // palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"]

var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

Funciones y Propiedades Básicas (IV)

- Funciones útiles para arrays:
  - *Length*(), calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
var numeroVocales = vocales.length; // numeroVocales = 5
```

  - *concat*(), se emplea para concatenar los elementos de varios arrays

```
var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6); // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

  - *join*(separador), es la función contraria a *split*(). Une todos los elementos de un *array* para formar una cadena de texto. Para unir los elementos se utiliza el carácter

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" "); // mensaje = "hola mundo"
```


Funciones y Propiedades Básicas (IV)

  - *pop*(), elimina el último elemento del *array* y lo devuelve. El *array* original se modifica y su longitud disminuye en 1 elemento

```
var array = [1, 2, 3];
var ultimo = array.pop(); // ahora array = [1, 2], ultimo = 3
```

  - *push*(), añade un elemento al final del *array*. El *array* original se modifica y aumenta su longitud en 1 elemento

```
var array = [1, 2, 3];
array.push(4); // ahora array = [1, 2, 3, 4]
```

Funciones y Propiedades Básicas (V)

- *shift*(), elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento

```
var array = [1, 2, 3];
var primero = array.shift(); // ahora array = [2, 3], primero = 1
```

- *unshift*(), añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento

```
var array = [1, 2, 3];
array.unshift(0); // ahora array = [0, 1, 2, 3]
```

- *reverse*(), modifica un array colocando sus elementos en el orden inverso a su posición original

```
var array = [1, 2, 3];
array.reverse(); // ahora array = [3, 2, 1]
```


Funciones y Propiedades Básicas (VI)

- Funciones útiles para números:
  - *NaN*, (del inglés, "Not a Number") JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división por cero)

```
var numero1 = 0;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

  - *isNaN*(), permite proteger a la aplicación de posibles valores numéricos no definidos

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
    alert("La división no está definida para los números indicados");
}
else {
    alert("La división es igual a => " + numero1/numero2);
}
```

Funciones y Propiedades Básicas (VI)

- *Infinity*, hace referencia a un valor numérico infinito y positivo (también existe el valor –*Infinity* para los infinitos negativos)

```
var numero1 = 10;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor Infinity
```

- *toFixed*(digitos), devuelve el número original con tantos decimales como los indicados por el parámetro dígitos y realiza los redondeos necesarios

```
var numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```


Funciones (I)

- Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Si una serie de instrucciones se repiten, se complica demasiado el código fuente de la aplicación:
  - El código de la aplicación es mucho más largo
  - Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción
  - Un trabajo muy pesado y propenso a cometer errores

Funciones (II)

- Las **funciones** son la solución a todos estos problemas, tanto en **JavaScript** como en el resto de lenguajes de programación. ***Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente***
- La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo ***"en este punto, se ejecutan las instrucciones que se han extraído"***

Funciones (III)

- Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en **JavaScript** se definen mediante la palabra reservada ***function***, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion([parametro1,parametro2,...]) {
    ...
    ...
    ...
    [return valor;]
}
```

Sentencias break y continue (I)

- Las sentencias ***break*** y ***continue*** permiten manipular el comportamiento normal de los bucles ***for*** para detener el bucle o para saltarse algunas repeticiones
  - La sentencia ***break*** permite terminar de forma abrupta un bucle
  - La sentencia ***continue*** permite saltarse algunas repeticiones del bucle

Otras Estructuras de Control (I)

- **Estructura *while*:** permite crear bucles que se ejecutan ninguna o más veces, dependiendo de la condición indicada. Su definición formal es:

```
while(condicion) {
...
}
```

- El funcionamiento del bucle ***while*** se resume en: *"mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle"*. Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta

Otras Estructuras de Control (II)

- **Estructura *do...while*:** es muy similar al bucle ***while***, salvo que en este caso siempre se ejecutan las instrucciones del bucle al menos la primera vez. Su definición formal es:

```
do {
...
} while(condicion);
```

- De esta forma, como la condición se comprueba después de cada repetición, la primera vez siempre se ejecutan las instrucciones del bucle. Es importante no olvidar que después del ***while*()** se debe añadir el carácter ;

Otras Estructuras de Control (III)

- **Estructura *switch*:** La estructura ***if...else*** se puede utilizar para realizar comprobaciones múltiples y tomar decisiones complejas. En los casos donde todas las condiciones dependen siempre de la misma variable, el código **JavaScript** resultante es demasiado redundante
- En estos casos, la estructura ***switch*** es la más eficiente, ya que está esta especializada en administrar de forma sencilla múltiples condiciones sobre la misma variable

Otras Estructuras de Control (IV)

- La definición formal de la estructura ***switch*** es:

```
switch(variable) {
    case valor_1:
        ...
        break;
    case valor_2:
        ...
        break;
    ...
    ...
    ...
    case valor_n:
        ...
        break;
    default:
        ...
}
```