JAVASCRIPT
Capítulo 3

Lenguajes de Marcas y Sistemas de Gestión de Información

Curso 2019/2020

Capítulo 3: PROGRAMACIÓN BÁSICA

1. Introducción
2. Variables
3. Tipos de Variables
4. Operadores
5. Estructuras de Control de Flujo
6. Funciones y Propiedades Básicas de JavaScript

1. INTRODUCCIÓN

Antes de comenzar a desarrollar programas y utilidades con JavaScript, es necesario conocer los elementos básicos con los que se construyen las aplicaciones.

Este capítulo te servirá para conocer la **sintaxis** específica de JavaScript.

2. VARIABLES

Una variable es un elemento que se emplea para **almacenar** y **hacer referencia** a otro valor. Gracias a las variables es posible crear programas que funcionan siempre igual independientemente de los valores concretos utilizados.

Las variables en los lenguajes de programación siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas.

Las variables en JavaScript se crean mediante la palabra reservada `var`.

```
var numero_1 = 3;

var numero_2 = 1;

var resultado = numero_1 + numero_2;
```

2. VARIABLES

La palabra reservada `var` solamente se debe indicar al definir por **primera vez** la variable, lo que se denomina declarar una variable.

Cuando se utilizan las variables en el resto de instrucciones del script, solamente es necesario indicar su nombre. En el ejemplo anterior sería un **error** indicar lo siguiente:

```
var numero_1 = 3;

var numero_2 = 1;

var resultado = var numero_1 + var numero_2;
```

2. VARIABLES

Una de las características 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`:

```
var numero_1 = 3;
var numero_2 = 1;
resultado = numero_1 + numero_2;
```

Como variable `resultado` **no está declarada, por lo que JavaScript crea una variable global** y le asigna el valor correspondiente:

```
numero_1 = 3;
numero_2 = 1;
resultado = numero_1 + numero_2;
```

En cualquier caso, **se recomienda declarar todas las variables que se vayan a utilizar.**

2. VARIABLES

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.

Identificadores Correctos

```
var $numero1;
var _$letra;
var $$$otroNumero;
var $_a__$4;
```

Identificadores Incorrectos

```
var 1numero;
var numero;1_123;
```

3. TIPOS DE VARIABLES

Aunque todas las variables de JavaScript se crean mediante la palabra reservada `var`, la forma en la que se les asigna un valor depende del **tipo de valor** que se quiere almacenar:

- Numéricas
- Cadenas de Texto
- Arrays
- Booleanos

3.1. VARIABLES 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:

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

3.2. CADENAS DE TEXTO

Se utilizan para **almacenar caracteres, palabras y/o frases de texto**. Para asignar el valor a la variable, 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';
```

El texto que se almacena en las variables no es tan sencillo. Si contiene comillas simples o dobles, se debe encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
/* 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';
```

3.2. CADENAS DE TEXTO

Existen **otros caracteres** que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.) Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una **combinación simple de caracteres**. A continuación se muestra la tabla de conversión que se debe utilizar:

Si se quiere incluir...	Se debe incluir...
Una nueva línea	`\n`
Un tabulador	`\t`
Una comilla simple	`\'`
Una comilla doble	`\"`
Una barra inclinada	`\\`

3.2. CADENAS DE TEXTO

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

El ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

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

3.3. ARRAYS

Un array es una **colección de variables**, que pueden ser todas del mismo tipo o cada una de un tipo diferente.

Para definir un array, se utilizan los caracteres `[` y `]` para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

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

*Si una aplicación necesita manejar los **días de la semana**, se pueden agrupar todas las variables relacionadas en una colección de variables o array:*

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

*Una **única variable** llamada* `dias` *almacena todos los valores relacionados entre sí, en este caso los días de la semana.*

3.3. ARRAYS

Una vez definido un array, es muy sencillo **acceder** a cada uno de sus elementos.

Cada elemento se accede indicando su posición dentro del array. La única complicación, que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el 0 y no en el 1:

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

Si queremos inicializar un array con X elementos lo podemos hacer de dos formas diferentes:

```
var arrayVacio = new Array(X);
var arrayVacio2 = [];
```

3.4. BOOLEANOS

Una variable de tipo boolean 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. También se conocen con el nombre de variables de tipo lógico.

Los **únicos valores** que pueden almacenar estas variables son `true` y `false`, por lo que no pueden utilizarse los valores verdadero y falso.

```
var clienteRegistrado = false;

var ivaIncluido = true;
```

4. OPERADORES

Las variables por sí solas son de poca utilidad. Para hacer programas realmente útiles, son necesarias otro tipo de herramientas.

Los operadores permiten **manipular el valor** de las variables, realizar **operaciones matemáticas** con sus valores y **comparar** diferentes **variables.** De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

- Asignación
- Incremento o Decremento
- Lógicos
- Matemáticos
- Relacionales

4.1. 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 = (igual):

```
var numero1 = 3;
```

A la **izquierda** del operador, siempre debe indicarse el nombre de una variable. A la **derecha** del operador, se pueden indicar variables, valores, condiciones lógicas, etc…

4.2. INCREMENTO O 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
```

El operador de incremento se indica mediante el **prefijo ++** en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad:

```
var numero = 5;
numero = numero + 1;
alert(numero);  // numero = 6
```

4.2. INCREMENTO O DECREMENTO

De forma equivalente, el operador decremento (indicado como un **prefijo --** en el nombre de la variable) se utiliza para **decrementar** el valor de la variable:

```
var numero = 5;

--numero;

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

El anterior ejemplo es equivalente a:

```
var numero = 5;

numero = numero - 1;

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

4.2. INCREMENTO O DECREMENTO

Los operadores de incremento y decremento no solamente se pueden indicar como prefijo del nombre de la variable, sino que también es posible utilizarlos como **sufijo**. En este caso, su comportamiento es similar pero muy diferente.

- Si el operador ++/-- se indica como **prefijo** del identificador de la variable, su valor se incrementa antes de realizar cualquier otra operación.
- Si el operador ++/-- se indica como **sufijo** del identificador de la variable, su valor se incrementa después de ejecutar la sentencia en la que aparece.

4.2. INCREMENTO O DECREMENTO

```
var numero1 = 5;

var numero2 = 2;

numeroA = numero1++ + numero2;

// numeroA = 7, numero1 = 6
```

En la instrucción `numeroA`, el valor de `numero1` se incrementa después de realizar la operación (primero se suma y `numeroA` vale 7, después se incrementa el valor de `numero1` y vale 6).

```
var numero1 = 5;

var numero2 = 2;

numeroB = ++numero1 + numero2;

// numeroB = 8, numero1 = 6
```

En la instrucción `numeroB`, en primer lugar se incrementa el valor de `numero1` y después se realiza la suma (primero se incrementa `numero1` y vale 6, después se realiza la suma y `numeroB` vale 8).

4.3. LÓGICOS

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que 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**.

- Negación
- AND
- OR

4.3. LÓGICOS: NEGACIÓN

Se utiliza para obtener el **valor contrario** al valor de la variable:

```
var visible = true;

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

La negación lógica se obtiene prefijando el **símbolo !** al identificador de la variable. El funcionamiento de este operador se resume en la siguiente tabla:

variable	!variable
`true`	`false`
`false`	`true`

4.3. LÓGICOS: AND

La operación lógica AND obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el **símbolo &&** y su resultado solamente es true si los dos operandos son true:

variable1	variable2	variable1 && variable2
true	true	true
true	false	false
false	true	false
false	false	false

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

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

4.3. LÓGICOS: OR

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el **símbolo ||** y su resultado es true si alguno de los dos operandos es true:

variable1	variable2	variable1 || variable2
true	true	true
true	false	true
false	true	true
false	false	false

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

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

4.4. MATEMÁTICOS

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

```
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
```

4.4. MATEMÁTICOS

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que es muy útil en algunas ocasiones. Se trata del operador "***módulo***", que calcula el resto de la división entera de dos números.

El operador módulo en JavaScript se indica mediante el **símbolo %**, que no debe confundirse con el cálculo del porcentaje:

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

numero1 = 9;
numero2 = 5;
resultado = numero1 % numero2; // resultado = 4
```

4.4. MATEMÁTICOS

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
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
```

4.5. RELACIONALES

Los operadores relacionales son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

El resultado de todos estos operadores siempre es un valor **booleano**:

```
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
```

4.5. RELACIONALES

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación. El **operador ==** se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// 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
```

4.5. RELACIONALES

Los operadores relacionales también se pueden utilizar con variables de tipo **cadena de texto**:

```
var texto1 = "hola";
var texto2 = "hola";
var texto3 = "adios";

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

Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) siguen un razonamiento **no intuitivo**: se compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto.

5. ESTRUCTURAS DE CONTROL DE FLUJO

Si los programas se hacen utilizando solamente variables y operadores, son una **simple sucesión lineal** de instrucciones básicas:

- No se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado.
- Tampoco se puede repetir de forma eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las estructuras de control de flujo:

- Estructura `IF`
- Estructura `IF…ELSE`
- Estructura `FOR`
- Estructura `FOR…IN`

5.1. ESTRUCTURA IF

Se emplea para tomar decisiones en función de una condición. Definición formal:

```
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:

```
var mostrarMensaje = true;

if(mostrarMensaje) {
        alert("Hola Mundo");
}
```

5.1. ESTRUCTURA IF

La condición que controla el `if()` puede **combinar** los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;
if(!mostrado) {
        alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores `AND` y `OR` permiten **encadenar varias condiciones** simples para construir condiciones complejas:

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

if(!mostrado && usuarioPermiteMensajes) {
        alert("Es la primera vez que se muestra el mensaje");
}
```

5.2. ESTRUCTURA IF...ELSE

Para este segundo tipo de decisiones, existe una estructura llamada `if...else`. Su definición formal es la siguiente:

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

Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

5.2. ESTRUCTURA IF...ELSE

Si la condición **se cumple**, se ejecutan todas las instrucciones que se encuentran dentro del `if()`. Si la condición **no se cumple**, se ejecutan todas las instrucciones contenidas en `else { }`.

```
var edad = 18;


if(edad >= 18) {
        alert("Eres mayor de edad");
}
else {
        alert("Todavía eres menor de edad");
}
```

5.3. ESTRUCTURA `FOR`

La estructura `for` permite realizar repeticiones o bucles de una forma muy sencilla. Su definición formal no es tan sencilla como la de `if()`:

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

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 condición.

- La **inicialización** es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La **condición** es el único elemento que decide si continua o se detiene la repetición.
- La **actualización** es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

5.3. ESTRUCTURA FOR

El ejemplo que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura `for`**:**

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


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

5.4. ESTRUCTURA `FOR...IN`

Es una estructura de control derivada de `for`. Su definición exacta implica el **uso de objetos**, que es un elemento de programación avanzada. Por tanto, solamente se va a presentar la estructura `for...in` adaptada a su uso en *arrays*. Su definición formal adaptada a los *arrays* es:

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

Es la más adecuada para recorrer *arrays*, ya que evita tener que indicar la inicialización y las condiciones del bucle `for` simple y funciona correctamente cualquiera que sea la longitud del *array*.

5.4. ESTRUCTURA FOR...IN

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]);
    }
```

6. FUNCIONES Y PROPIEDADES BÁSICAS DE JAVASCRIPT

JavaScript incorpora una serie de herramientas y utilidades para el **manejo de las variables**. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

- Funciones para Cadenas de Texto
- Funciones para Arrays
- Funciones para Números

6.1. FUNCIONES PARA CADENAS DE TEXTO

- `length`: **calcula la longitud de una cadena de texto (el número de caracteres que la forman)**

```
var mensaje = "Hola Mundo";

var numeroLetras = mensaje.length; // numeroLetras = 10
```

- `+`: **se emplea para concatenar varias cadenas de texto**

```
var mensaje1 = "Hola";

var mensaje2 = " Mundo";

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

6.1. FUNCIONES PARA CADENAS DE TEXTO

- Además del operador +, también se puede utilizar la función `concat()`

```
var mensaje1 = "Hola";

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

- Los **espacios en blanco** se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```
var mensaje1 = "Hola";

var mensaje2 = "Mundo";

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

6.1. FUNCIONES PARA CADENAS DE TEXTO

- `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"
```

6.1. FUNCIONES PARA CADENAS DE TEXTO

- `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
```

- `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 hasta el final:**

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5);     // porcion = "Mundo"
porcion = mensaje.substring(0, 5);     // porcion = "Hola "
```

6.2. FUNCIONES 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([7, 8, 9]); // array3 = [1, 2, 3, 7, 8, 9]
```

6.2. FUNCIONES PARA *ARRAYS*

- `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]
```

6.2. FUNCIONES PARA *ARRAYS*

- `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]
```

6.2. FUNCIONES PARA *ARRAYS*

- `splice(x,numElementos)`: **elimina** un número de **elementos** desde una posición X del array y lo devuelve. El array original se ve modificado y su longitud disminuida en *numElementos*.

```
var array = [“a”, “b”, “c”, ”d”, ”e”];
array.splice(2,2);
// ahora array = [“a”, “b”, “e”]
```

6.2. FUNCIONES PARA *ARRAYS*

- `join(separador)`: **Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado**

```
var array = ["hola", "mundo"];

var mensaje = array.join(""); // mensaje = "holamundo"

mensaje = array.join(" ");    // mensaje = "hola mundo"
```

- `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]
```

6.3. FUNCIONES PARA NÚMEROS

- `NaN`: **indica un valor numérico no definido** (por ejemplo, la división 0/0).

```
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(”División no definida para los números indicados");
}
else {
        alert("La división es igual a => " + numero1/numero2);
}
```

6.3. FUNCIONES PARA NÚMEROS

- `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 *digitos*** y realiza los redondeos necesarios:

```
var numero1 = 4564.34567;

numero1.toFixed(2); // 4564.35

numero1.toFixed(6); // 4564.345670

numero1.toFixed(); // 4564
```