UNIDAD 3 Primeras interacciones

Variables y tipos de datos

Variables

Las variables son elementos que pueden tener un valor que se pueda modificar a lo largo de nuestro *script*, sea este texto, un número, un valor *booleano* o un *array* (listado de valores).

Una variable se declara con la instrucción **`var`** y el nombre que queramos (que no empiece por números, tenga guiones o espacios, o sea una palabra restringida de JavaScript).

Ejemplo: **`var edadUsuario;`**

Si sabemos el valor inicial de una variable, podemos asignarlo al mismo tiempo que la declaramos: **`var edadUsuario = 20;`**

Además, podemos hacer una declaración múltiple de variables separadas por comas.

Ejemplo:

```
var edadUsuario = 20, nombreUsuario = "Laura", socio = true;
```

Palabras reservadas en la nomenclatura de variables

Las siguientes palabras están reservadas por JavaScript y no se pueden usar como variables (porque ya se están usando o porque se podrían usar en el futuro):

```
abstract, boolean, break, byte, case, catch, char, class, const,
continue, debugger, default, delete, do, double, else, enum,
export, extends, false, final, finally, float, for, function,
goto, if, implements, import, in, instanceof, int, interface,
long, native, new, null, package, private, protected, public,
return, short, static, super, switch, synchronized, this, throw,
throws, transient, true, try, typeof, var, void, volatile, while,
with
```

Alcance de las variables

Las variables normalmente están acotadas a la función en la que se encuentran (variables locales). Si no se encuentran dentro de ninguna función, se tratará de variables globales. Cualquier función puede acceder y manipular las variables globales, pero a las variables locales normalmente solo se puede acceder desde su propio ámbito.

Numerales

Los valores numerales se escriben únicamente con cifras y sin usar comillas. Pueden interpretarse y manipularse directamente mediante operadores.

Literales

Los valores literales o *strings* son cadenas de texto. Estos valores siempre se escriben entrecomillados porque JavaScript no los puede interpretar directamente (los confundiría con variables).

Booleanos

Los valores *booleanos* solo pueden ser **`true`** o **`false`**, es decir, verdadero o falso. Los condicionales emplean mucho estos valores, ya que ejecutarán o no un conjunto de líneas de código en función de si una afirmación es verdadera o falsa (por ejemplo, si el valor de una variable es igual a un valor concreto). El valor **`true`** o **`false`** puede ser interpretado por JS y por tanto no va entrecomillado.

Arrays

Los *arrays* son listados de varios valores, separados por comas. El conjunto de todos estos valores se agrupa mediante corchetes `[]`.

Ejemplo: **`var coloresDisponibles = ["rojo", "azul", "amarillo"];`**

`undefined`

El valor **`undefined`** indica que la variable o función a la que nos referimos no ha sido definida. Una función también puede devolver **`undefined`** al ser ejecutada si no tiene la instrucción de devolver ningún valor concreto (lo cual no es necesariamente un error).

`null`

El valor **`null`** representa un valor nulo o vacío. Por ejemplo, cuando hacemos referencia a una variable que ha sido declarada, pero no le hemos asignado ningún valor.

`NaN`

El valor **`NaN`** indica que el elemento al que hacemos referencia no es un número (*Not-A-Number*). Es un resultado que podemos obtener si intentamos restar al literal "Hola" el numeral 5: como el primero no es un número, no puede realizar la operación aritmética y devuelve el valor **`NaN`**.

Operador de asignación y operadores aritméticos

Operador de asignación: `=`

El operador **=** permite asignar un valor a una variable. Para comparar dos valores, usaremos el comparador **==**.

Operador aritmético de adición: `+`

El operador **+** permite sumar valores numéricos o concatenar *strings* (literales). Es el único operador aritmético que podremos usar con texto.

Operador aritmético de sustracción: `-`

Permite hacer una resta entre dos valores numéricos.

Operador aritmético de división: `/`

Divide dos valores numéricos.

Operador aritmético de multiplicación: `*`

Multiplica dos valores numéricos.

Operador aritmético de incremento: `++`

Podemos añadir el operador **++** al final del nombre de una variable para incrementar en 1 su valor. Una variable **`"ejemplo"`** que tenga un valor de **`3`**, subirá a **`4`** si escribimos **`ejemplo++;`**.

Operador aritmético de decremento: `--`

Podemos añadir el operador **--** al final del nombre de una variable para reducir en 1 su valor. Una variable **`"ejemplo"`** que tenga un valor de **`3`**, bajará a **`2`** si escribimos **`ejemplo--;`**.

Operador aritmético de módulo: `%`

Permite obtener el resto de una división de dos números (por ejemplo, **`10 % 3`** daría como resultado **`1`**).

Operadores de comparación

Operador de comparación: ==
Permite comprobar si dos valores son iguales.

Operador de comparación: !=
Permite comprobar si dos valores son diferentes.

Operador de comparación: ===
Podemos usar `===` en vez de `==` si queremos comprobar no solo que dos valores son iguales, sino que son estrictamente del mismo tipo (por ejemplo, no es lo mismo el valor numérico `3` que el valor literal `"3"`, pero `==` diría que sí es lo mismo, mientras que `===` no).

Operador de comparación: !==
Al igual que `===`, permite comparar dos valores estrictamente; en este caso si el valor y/o el tipo de variable son diferentes.

Operador de comparación: >
Permite saber si un valor es mayor que otro.

Operador de comparación: <
Permite saber si un valor es menor que otro.

Operador de comparación: >=
Permite saber si un valor es mayor o igual que otro.

Operador de comparación: <=
Permite saber si un valor es menor o igual que otro.

Operador lógico AND: &&
Permite saber si dos expresiones se cumplen. Se usa habitualmente en condicionales.

Por ejemplo: `((5 > 2) && (2 <= 3))`

Operador lógico OR: ||
Permite saber si al menos una de dos expresiones se cumplen. Se usa habitualmente en condicionales.

Por ejemplo: `((4 > 2) || (1 > 6))`

Operador lógico NOT: `!`

Permite saber si una expresión no se cumple. Se puede poner al principio del nombre de una variable para preguntar si su valor es false o no existe.

Por ejemplo: `!(4 < 2)`

Funciones

Declarar una función

Una función nos permite agrupar código entre `{}` para poderlo reutilizar o llamar cuando es necesario. Normalmente le asignaremos un nombre y podemos pasarle argumentos a través del paréntesis.

Ejemplo:

```
function escribeConsola(mensaje){
  console.log(mensaje);
}
```

Ejecutar una función

Para ejecutar una función, escribiremos su nombre junto a los paréntesis. Si queremos pasarle argumentos (valores que serán interpretados dentro de la función), lo podemos hacer dentro del paréntesis.

Ejemplo:

```
escribeConsola("Hola, ¿qué tal?");
```

Funciones anónimas

En ocasiones necesitamos agrupar código en una función por necesidades sintácticas, pero queremos que el código se ejecute directamente sin tener que llamar a la función más adelante. Si es el caso, podemos declarar una función anónima, que se declara igualmente con la palabra function, pero sin ponerle un nombre, solo con paréntesis.

Ejemplo:

```
function () {
  console.log("Hola");
}
```

`return …;`

La palabra **`return`** fuerza la salida de la función que se está ejecutando, dejando de interpretar las líneas a continuación. Se puede introducir un valor a continuación de la palabra si la función que se ha llamado debe "devolver" un resultado.

Event listeners

`document.addEventListener("evento", función)`

Podemos asociar una función a un evento concreto con el método **`addEventListener()`** del objeto documento. Los dos argumentos que espera este método son el tipo de evento (**`"click"`**, **`"submit"`**, etc, entrecomillados) y la función que se ejecutará (puede ser una función declarada previamente o una función anónima declarada dentro del propio paréntesis). El nombre de la función asociada se escribe sin paréntesis (salvo que sea una función anónima) para no provocar su ejecución antes de tiempo.

`evento.preventDefault()`

Podemos cancelar la acción por defecto de un elemento (enviar un formulario, seguir un enlace...) si recogemos el evento como argumento de la función que llamamos en un listener y luego ejecutamos el método **`preventDefault()`** en este mismo argumento.

Ejemplo:

```
function recogeDatos(evento) {
  evento.preventDefault();
}
```

`this`

La palabra **`this`** hace referencia al elemento que ha activado un listener, y es bastante útil para recuperar datos. Por ejemplo, si asociamos una función a un evento de *click*, podríamos utilizar la palabra **`this`** para referirnos al elemento en el que hemos hecho *click*.

Acceder al DOM

`document.querySelector() y document.querySelectorAll()`

Podemos seleccionar un nodo dentro del DOM con el método **`querySelector()`**. Podemos aplicarlo sobre todo el documento como **`document.querySelector()`**, pero también podríamos aplicarlo sobre un nodo seleccionado previamente, que hayamos guardado en una variable. El argumento que podemos pasar a este método es un selector de CSS, entrecomillado. El método **`querySelector()`** devolverá el primer nodo que cumpla con el selector indicado. Si queremos seleccionar todos los nodos que cumplan con el selector indicado, deberemos usar el método **`querySelectorAll()`**, y lo que obtendremos como resultado es un listado de nodos (**`NodeList`**), que funciona de una manera equivalente a los *arrays*.

`elemento.textContent`

Podemos recuperar o modificar el valor textual de un nodo con **`textContent`**. Con este método no podemos introducir etiquetas HTML, se visualizarán como texto escrito.

Ejemplo:

```
document.querySelector("h1").textContent = “Bienvenido";
```

`elemento.innerHTML`

Podemos recuperar o modificar el contenido HTML de un nodo con **`innerHTML`**.

Ejemplo:

```
document.querySelector(".destacado").innerHTML =
“<p>Importante</p>";
```

`elemento.value`

Podemos recuperar el valor de un input de formulario accediendo a su propiedad **`value`** (casi todos los elementos son objetos para JavaScript, y tienen propiedades a las que podemos acceder).

Ejemplo:

```
var nombre = document.querySelector("#nombre").value;
```

Condicionales

Condicionales

Los condicionales permiten agrupar código y que solo se ejecute si se dan ciertas condiciones. El condicional más popular es **`if…else`**. La sintaxis de este condicional es la siguiente:

```
if (condición) {
  resultado;
}
```

La condición que escribimos entre paréntesis suele incorporar un operador de comparación (si un valor es igual a otro valor, o mayor, o menor, etc). A continuación, podemos (opcionalmente) escribir otra condición, si la primera no se ha cumplido, para lo que usaremos **`else if`** (condición). De manera opcional también, podríamos escribir finalmente un **`else`**, que implicaría el código a ejecutar si ninguna de las condiciones anteriores aplica.

Ejemplo:

```
if(edadUsuario > 17){
     console.log("es mayor de edad");
}
else if(edadUsuario < 12){
     console.log("es un niño");
}
else{
     console.log("es un adolescente");
}
```