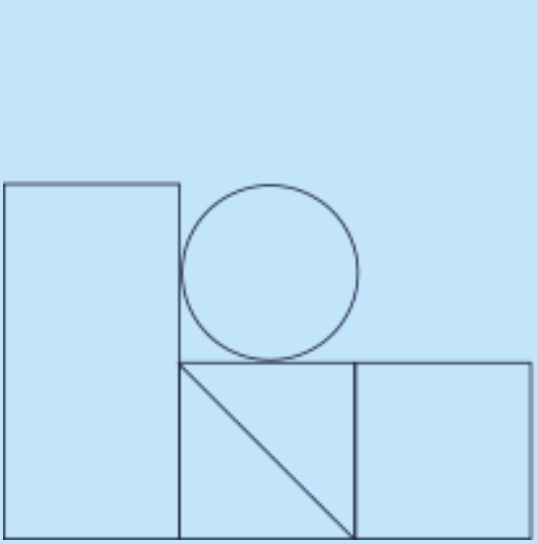
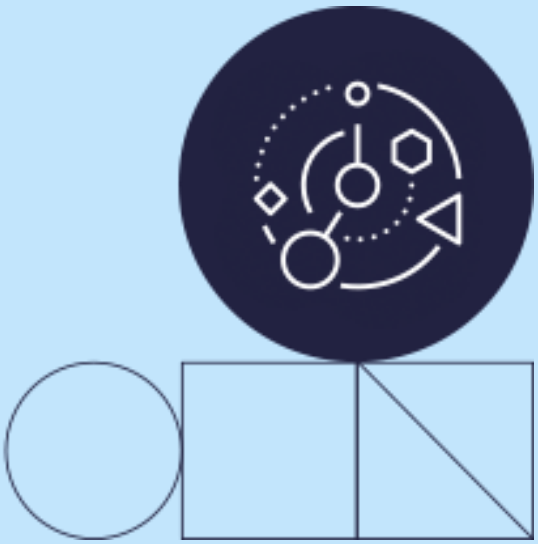
IBM SkillsBuild

IBM

Conceptos básicos de programación

Definición de variable

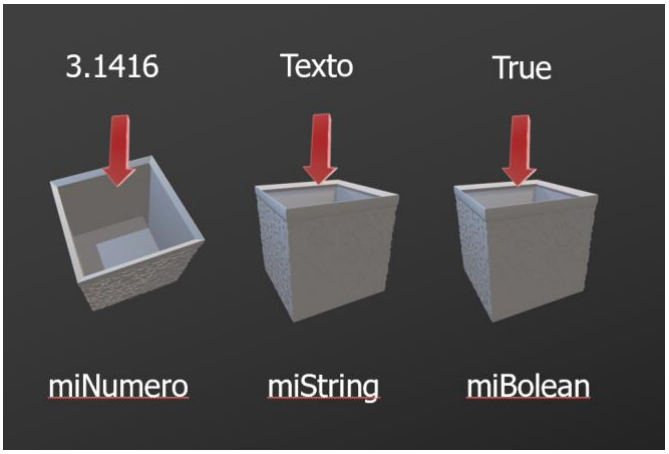




Índice

Introducción

Una variable es un espacio de memoria donde se almacenará un valor, que podrá cambiar a lo largo del programa. Algo así como una “caja” donde guardaremos datos:



Reglas para definir una variable.

Estas reglas son comunes a la mayoría de los lenguajes de programación:

- Debe comenzar por una letra, el símbolo **$** ó el símbolo **_**
- Sólo puede contener letras, números, el símbolo **$** ó el símbolo **_**
- Las variables no pueden tener espacios en blanco.
- Se pueden usar tildes, pero no es recomendable. Tampoco es recomendable usar la “**ñ**”.
- En algunos lenguajes existe distinción entre mayúsculas y minúsculas. Son lo que se denomina lenguajes *Case sensitive*.
- Como norma general se ponen en minúsculas.
- No pueden ser palabras reservadas.

Tipos de datos

Como vimos en la lección anterior los siguientes tipos de datos primitivos que poseen la mayoría de los lenguajes de programación de alto nivel.

Vamos a centrarnos en el lenguaje Javascript, que es objeto de este curso.

Más adelante veremos detalladamente los tipos de datos en Javascript, pero sirva como pincelada.

Javascript nos proporciona una serie de datos primitivos con los que poder trabajar, como son:

- **Number**: Numéricos, ya sean enteros o de punto flotante (3.4 , 2.5...etc).
  En este lenguaje, a la hora de declarar variables, no se hacen distinciones entre números enteros y decimales. Ya se encarga Javascript de decidir si son de uno u otro tipo según el dato que introduzcamos en nuestra variable.
- **String**: Cadenas de caracteres, es decir, texto. Ya sean letras, palabras o frases.
- **Boolean**: Boleanos, es decir **true** o **false**.

Hay veces que puede ser necesario conocer el tipo de una variable en un momento dado. Para ello, JavaScript proporciona un operador que, al aplicarlo sobre una determinada variable, puede obtener el tipo de dato que ahora mismo está asignado a la variable. Este operador es *typeof():*

```
typeof(nombreVariable);
```

Si una variable ha sido declarada pero no iniciada, es decir, no le hemos dado ningún valor, su valor es *undefined*. Su valor es desconocido. No es que sea cero, es desconocido.

Tipos complejos

Son creados por el usuario a partir de estructuras del lenguaje y/o agrupaciones de elementos de tipo simple. Así, es posible encontrar el tipo de dato función (*function*), que permite asignar a una variable una función (las funciones se ven con posterioridad), y el tipo de dato objeto, que engloba un conjunto de posibles tipos de datos (arrays o tablas, por ejemplo, que también se ven más adelante, junto con el propio tipo *Object*, que hacer referencia a un objeto genérico).

Declaración de variables

Las palabras clave de JavaScript son palabras reservadas. Las palabras reservadas no se pueden utilizar como nombres de variables.

Las variables no declaradas son siempre globales, y no existen hasta que se ejecuta el código que las asigna. Una variable no declarada es configurable (por ejemplo, se puede borrar).

Se recomienda siempre declarar las variables a utilizar, ya sean locales o globales. Cuando no declaramos una variable, JS la crea en el Window Object. Llegando incluso a reasignar el valor de esta sin pedirnos permiso, provocando un comportamiento inesperado.

Lo primero que debemos hacer para usar una variable es “definirla“, (crearla):

```
var numero;
```

Ya tenemos nuestra variable creada, pero no le hemos dado ningún valor.

Para poder trabajar con ella hay que “inicializarla”, es decir, darle un valor inicial.

Para darle un valor:

```
numero = 5;
```

Una vez que ya tenemos la variable definida, para darle un valor no necesitamos usar la palabra reservada *var*.

También podemos **definir** e **inicializar** una variable en un solo paso:

```
var numero = 5;
```

Se pueden definir o inicializar varias variables en una sola línea, pero se desaconseja:

```
var nombre, apellido, DNI;

var nombre="Angel", apellido="García", DNI="12345678A";
```

Una variable declarada pero no iniciada tendrá como valor *undefined*.

Constantes

Se define constante como un espacio en la memoria del ordenador cuyo valor **no** podrá variar a lo largo de la ejecución del programa. Es decir, una "variable" cuyo valor no varía nunca.

Por convenio, los nombres de las constantes van en mayúsculas.

Sintaxis:

```
const NOMBRE = "Angel";
```

Una constante no puede compartir su nombre con una función o variable en el mismo ámbito.

Scope, ámbito

El ámbito (scope) de una variable es el contexto (la zona del programa) en la que la variable es visible /accesible (está declarada), y por tanto puede ser usada. El scope determina la accesibilidad de nuestro código, es decir, desde dónde podemos acceder a nuestras variables.

Tenemos dos scopes para las variables:

- **Global**: Podemos acceder a ellas desde cualquier parte de nuestro código.
- **Local**: Podemos acceder a ellas sólo dentro del ámbito en que fueron definidas. Normalmente funciones. Serán accesibles desde la propia función o desde funciones anidadas en niveles superiores a la anterior. Es decir, las variables locales se pueden ver desde dentro hacia fuera, pero nunca desde fuera hacia dentro de la función.

Las variables locales se definen dentro de una función y solo se tiene acceso dentro de la misma función. Una variable local se la define antecediendo la palabra clave *let* al nombre de la variable.

Si intentamos acceder a la variable fuera de la función en la que se declaró, se produce un error.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {
    var mensaje = "Mensaje de prueba";
  }
  creaMensaje();
  alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada *creaMensaje* que crea una variable llamada mensaje. A continuación, se ejecuta la función mediante la llamada *creaMensaje()*; y seguidamente, se muestra mediante la función *alert()* el valor de una variable llamada mensaje.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable mensaje se ha definido dentro de la función *creaMensaje()* y por tanto, es una variable local que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje. De esta forma, para mostrar el mensaje en el código anterior, la función alert() debe llamarse desde dentro de la función *creaMensaje()*:

```
function creaMensaje() {
    var mensaje = "Mensaje de prueba";
    alert(mensaje);
  }
  creaMensaje();
```

Además de variables locales, también existe el concepto de **variable global**, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = "Mensaje de prueba";

function muestraMensaje() {
  alert(mensaje);
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función *muestraMensaje()* se quiere hacer uso de una variable llamada mensaje y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable mensaje.

El motivo es que en el código JavaScript anterior, la variable mensaje se ha definido fuera de cualquier función. Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada mensaje, la variable global creada anteriormente permite que la instrucción *alert()* dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada *var* o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante *var* se consideran locales y las variables que no se han declarado mediante *var*, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada *var*, para que se transforme en una variable global:

```
function creaMensaje() {
    mensaje = "Mensaje de prueba";
  }

  creaMensaje();
  alert(mensaje);
```

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";

function muestraMensaje() {
  var mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:

- gana la de fuera
- gana la de dentro
- gana la de fuera

Dentro de la función, la variable local llamada mensaje tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe? En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
  mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

En este caso, los mensajes mostrados son:

- gana la de fuera
- gana la de dentro
- gana la de dentro

La recomendación general es definir como variables locales todas las variables que sean de uso exclusivo para realizar las tareas encargadas a cada función. Las variables globales se utilizan para compartir variables entre funciones de forma sencilla.

Todo esto se verá con mucho más detenimiento cuando demos las variables en el módulo de Javascript.