UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

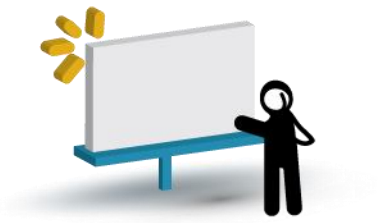
Desarrollo Web con Javascript

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

UTN.BA UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL BUENOS AIRES | Centro de e-Learning

Módulo 1: Programando con Javascript

Unidad 2: Variables y operadores



Presentación:

JavaScript, es un lenguaje de programación de páginas web de lado del cliente, esto significa, que cuando estamos viendo una página que utiliza JavaScript, hemos descargado el código a nuestro navegador y nuestro navegador lo está ejecutando de acuerdo con las acciones realizadas en la página.

Actualmente es una pieza fundamental en el desarrollo de aplicaciones web y es usado por las grandes compañías como Google, Yahoo, Microsoft, etc y se ha convertido en una herramienta tan poderosa, que su conocimiento es uno de los puntos más valorados en las búsquedas laborales de desarrolladores web.

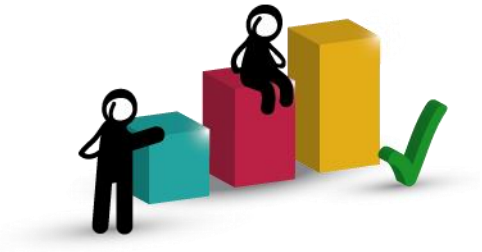
Objetivos:

Que los participantes:

- Logren conocer los conceptos básicos del lenguaje

Bloques temáticos:

1. Variables
2. Ámbito de las variables
3. Literales
4. Comentarios
5. Mayúsculas y minúsculas
6. Separación de instrucciones
7. Operadores



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

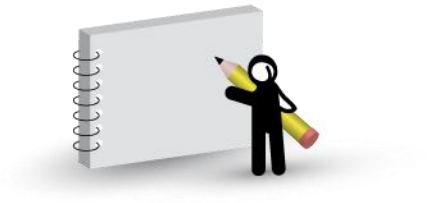
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Variables

Para comenzar debemos observar que a diferencia de otros lenguajes de alto nivel en los que se necesita definir variables e indicar el tipo antes de poder utilizarlas, Javascript es más flexible en este aspecto al permitir crear variables cuando estas sean necesarias.

Una variable es un elemento definido por el usuario que contendrá algún valor de cualquier tipo. Por valores nos referimos a un valor numérico, un texto, un carácter o cualquier otra cosa definida por el usuario. Esto significa que no es necesario especificar el tipo de información que contendrá una variable al momento de ser creada. De esta forma, podemos asignar un dato a una variable de diferente tipo dependiendo de nuestros requerimientos.

Esto, por un lado, nos da una herramienta flexible y poderosa, pero al mismo tiempo, nos obliga a ser consistentes al momento de definir el uso que le daremos a cada variable.

Las variables almacenan y recuperan datos, llamados “valores”. Una variable puede referirse a un valor que cambia o se cambia. Las variables son referenciadas por su nombre, y el nombre que se les asigna debe ser conforme a ciertas reglas:

- Debe empezar con una letra o un guion bajo _
- Los caracteres siguientes pueden ser números (0-9), letras mayúsculas o minúsculas

Para definirlas se les deberá anteponer la palabra reservada “var” (aunque no es condicionante).

Si utilizamos la palabra var, estaremos definiendo una variable local, que es una variable específica para un método, función, etc. Es decir, si declaramos la variable dentro de una función, a esta variable no podremos acceder desde otra función.

Si no utilizamos la palabra var, estaremos definiendo una variable global, que es una variable general para todo el código, es decir, podemos utilizar esta variable en todas las funciones, métodos, etc. de este código.

Ejemplos de definiciones erróneas:

```
var Mis Productos = ‘Computadoras’
```

No podemos declarar una variable con espacios en su nombre

```
var 1erproducto = 'Monitor'
```

No podemos declarar una variable que comience con un número

```
var $producto = 'Teclado'
```

No podemos declarar una variable que empiece con un símbolo

Ejemplo de definiciones correctas

```
var Mis_Productos = 'Computadora'
var primerProducto = 'Monitor'
var producto1 = 'Teclado'
```

Las variables en Javascript pueden ser de alcance local o global. Una variable global es accesible, es decir, se puede utilizar, desde cualquier <script> de la página mientras que una variable local solo lo es desde la función en la que fue declarada.

Podemos crear una variable global asignándole un valor:

```
Variable = 5
```

Sin embargo, si estamos codificando dentro de una función y queremos crear una variable local que solo tenga alcance dentro de esa función, debemos declarar la nueva variable haciendo uso de la palabra reservada "var"

```
function nuevaFuncion () {
var variable = 1
}
```

Aunque en Javascript no es estrictamente necesario declarar una variable antes de utilizarla, la declaración mediante el uso de la palabra reservada "var" es siempre recomendable, por claridad.

Ámbito de las variables

Se le llama ámbito de las variables al lugar donde estas están disponibles.

Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como Javascript se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella.

De este modo, no podremos acceder a variables que hayan sido definidas en otra página. Este es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página, aunque no será el único, ya que también podremos declarar variables en lugares más acotados.

Variables globales

Como hemos dicho, las variables **globales** son las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web. Para declarar una variable global a la página simplemente lo haremos en un script, con la palabra var.

```
<script>

var variableGlobal

</script>
```

Las variables globales son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos, como el onclick, que se puede incluir dentro de determinadas etiquetas HTML.

Variables locales

También podremos declarar variables en lugares más acotados, como por ejemplo una función. A estas variables les llamaremos **locales**.

Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
<script>
function miFuncion (){
 var variableLocal
}
</script>
```

En el script anterior hemos declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función (más adelante profundizaremos sobre el uso de funciones).

Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito. No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está ocupado por la local y es ella quien tiene validez.

En resumen, la variable que tendrá validez en cualquier sitio de la página es la global. Menos en el ámbito donde está declarada la variable local, que será ella quien tenga validez.

```
<script>
var numero = 2
function miFuncion (){
 var numero = 19
 document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
</script>
```

Un consejo para los principiantes podría ser no declarar variables con los mismos nombres, para que nunca haya lugar a confusión sobre qué variable es la que tiene validez en cada momento.

Diferencias entre utilizar var o no

Como hemos dicho, en Javascript tenemos libertad para declarar o no las variables con la palabra var, pero los efectos que conseguiremos en cada caso serán distintos.

En concreto, cuando utilizamos var estamos haciendo que la varible que estamos declarando sea local al ámbito donde se declara. Por otro lado, si no utilizamos la palabra var para declarar una variable, ésta será global a toda la página, sea cual sea el ámbito en el que haya sido declarada.

En el caso de una variable declarada en la página web, fuera de una función o cualquier otro ámbito más reducido, nos es indiferente si se declara o no con var, desde un punto de vista funcional.

Esto es debido a que cualquier variable declarada fuera de un ámbito es global a toda la página. La diferencia se puede apreciar en una función por ejemplo, ya que si utilizamos var la variable será local a la función y si no lo utilizamos, la variable será global a la página.

Esta diferencia es fundamental a la hora de controlar correctamente el uso de las variables en la página, ya que si no lo hacemos en una función podríamos sobreescribir el valor de una variable, perdiendo el dato que pudiera contener previamente.

```
<script>
var numero = 2
function miFuncion (){
numero = 19
 document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
```

```
//llamamos a la función
miFuncion()
document.write(numero) //imprime 19
</script>
```

En este ejemplo, tenemos una variable global a la página llamada numero, que contiene un 2. También tenemos una función que utiliza la variable numero sin haberla declarado con var, por lo que la variable numero de la funcion será la misma variable global numero declarada fuera de la función.

En una situación como esta, al ejecutar la función se sobreescribirá la variable numero y el dato que había antes de ejecutar la función se perderá.

Tipos de variables

Cuando declaramos una variable en Javascript no es necesario determinar el tipo de dato a almacenar, sin embargo, de acuerdo al contenido, podemos identificar distintos tipos de variables:

Numéricas

Se utilizan para almacenar valores numéricos enteros (llamados integer en inglés) o decimales (llamados float en inglés).

En este caso, 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 descuento = 65;       // variable tipo entero
var total = 652.53;       // variable tipo decimal
```

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';
var letraSeleccionada = 'c';
```

En ocasiones, el texto que se almacena en las variables no es tan sencillo. Si por ejemplo el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de 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';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, 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	\\

De esta forma, 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";
```

Este mecanismo de JavaScript se denomina "**mecanismo de escape**" de los caracteres problemáticos, y es habitual referirse a que los caracteres han sido "**escapados**".

Booleanos

Las variables de tipo boolean o booleano también se conocen con el nombre de variables de tipo lógico.

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.

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

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

Arrays

También podemos trabajar con arrays, también denominados vectores, matrices o arreglos.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:

```
var dia1 = "Lunes";
var dia2 = "Martes";
var dia3 = "Miércoles";
var dia4 = "Jueves";
var dia5 = "Viernes";
var dia6= "Sábado";
var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array. El ejemplo anterior se puede rehacer de la siguiente forma:

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

Ahora, una única variable llamada **dias** almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los

caracteres [corchetes] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

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

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"
```

Veremos los arrays o vectores con mayor profundidad en la próxima unidad.

Literales

Los literales son valores que se utilizan para inicializar variables, es decir, para dar los valores que inicialmente se desean que tengan las variables. Por ejemplo:

var Numero = 0

var Logica = true

var Texto = “”

Estos tipos de valores, denominados literales, son expresiones constantes de valores para tipos de datos. Al mismo tiempo que se dan valores iniciales a las variables, son de utilidad pues nos ayudan a reconocer el tipo de dato que se ha asignado a cada variable.

A continuación se explicarán los distintos tipos de literales:

- Valores literales para números:
    - Números enteros: Pueden utilizarse 3 bases distintas para números enteros:
        - La decimal, base 10

          Se emplea una serie de dígitos sin ceros delante, por ejemplo: 23
        - La hexadecimal, base 16

          Se antepone al número 0x. Incluyen dígitos del 0 al 9 y letras de la A a la F (como los colores en CSS). Por ejemplo, 0xFF
        - La octal, base 8

          Se antepone un cero al número. Los literales en base octal solo pueden incluir los dígitos del 0 al 7. Por ejemplo, 010
- Valores literales para booleanos
    - Se especifican mediante verdadero o falso
        - Ejemplo: true o false
- Valores literales para cadenas de texto

- Se especifican empleando cero o más caracteres dentro de comillas doble o simples
  - “nombre”
  - “”
- Los literales para cadenas de texto pueden incluir también caracteres especiales, por ejemplo:
  - “hola \r chau”
  - Lista de caracteres especiales:
    - \b ➔ Retroceso
    - \f ➔ Salto de página
    - \r ➔ Salto de línea
    - \t ➔ Tabulador

Comentarios

Es recomendable documentar el documento de Javascript con comentarios, pues además de hacer un programa más fácil de entender, será útil para una posterior modificación. Si nos acostumbramos a poner comentarios referentes a qué hace el programa, y cómo lo hace, no perderemos tiempo en revisiones posteriores. También se pueden utilizar los comentarios para poner algunas notas referentes al autor del código.

Javascript permite 2 formas de comentarios:

- Comentarios de una línea, precedidos por //. Por ejemplo:

```
// Esto es un comentario de una línea
```

- Comentarios de más de una línea, cerrados por /* */. Por ejemplo:

```
/* Este comentario
Tiene más de una línea */
```

Mayúsculas y minúsculas

En javascript se han de respetar las mayúsculas y las minúsculas.

Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error de sintaxis.

Por convención los nombres de los elementos se escriben en minúsculas, salvo que se utilice un nombre con más de una palabra, pues en ese caso se escribirán con mayúsculas las iniciales de las palabras siguientes a la primera.

Por ejemplo:

```
var primerProducto
```

Separación de instrucciones

Las distintas instrucciones que contienen nuestros scripts se han de separar convenientemente para que el navegador no indique los correspondientes errores de sintaxis.

Javascript tiene dos maneras de separar instrucciones.

- La primera es a través del carácter punto y coma (;)
- La segunda es a través de un salto de línea.

Por esta razón Las sentencias Javascript no necesitan acabar en punto y coma a no ser que coloquemos dos instrucciones en la misma línea.

Por ejemplo:

Utilizando punto y coma (;)

```
var nombreUsuario = prompt(“Indica tu nombre”); var apellidoUsuario = prompt (“Indica tu apellido”)
```

Utilizando salto de línea

```
var nombreUsuario = prompt(“Indica tu nombre”)

var apellidoUsuario = prompt (“Indica tu apellido”)
```

Operadores

Al desarrollar programas en cualquier lenguaje se utilizan los operadores.

Éstos sirven para hacer los cálculos y operaciones necesarios para llevar a cabo sus objetivos.

Un programa que no realiza operaciones solo se puede limitar a hacer siempre lo mismo, es el resultado de estas operaciones lo que hace que un programa varíe su comportamiento según los datos que obtenga.

Existen operaciones más sencillas o complejas, que se pueden realizar con operandos de distintos tipos de datos, como números o textos, veremos en este capítulo de manera detallada todos estos operadores.

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 = (no confundir con el operador == que se verá más adelante y sirve para comparar):

```
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:

```
var numero1 = 3;
var numero2 = 4;
```

Incremento y 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.

Ejemplo:

```
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. Por tanto, el anterior ejemplo es equivalente a:

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

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
```

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

Uno de los operadores lógicos más utilizados es el de la 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.

Si la variable original es de tipo booleano, es muy sencillo obtener su negación. Sin embargo, ¿qué sucede cuando la variable es un número o una cadena de texto? Para obtener la negación en este tipo de variables, se realiza en primer lugar su conversión a un valor booleano:

- **Si la variable contiene un número**, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).
- **Si la variable contiene una cadena de texto**, se transforma en false si la cadena es vacía ("") y en true en cualquier otro caso.

```
var cantidad = 0;
vacio = !cantidad;  // vacio = true

cantidad = 2;
vacio = !cantidad;  // vacio = false

var mensaje = "";
mensajeVacio = !mensaje;  // mensajeVacio = true

mensaje = "Bienvenido";
mensajeVacio = !mensaje;  // mensajeVacio = false
```

AND

La operación lógica **AND** (y) 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 (verdaderos):

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

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

OR

La operación lógica **OR** (o) 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:

Matemáticos

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

```
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
```

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones.

Se trata del operador "módulo", que calcula el resto de la división entera de dos números. Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2. El resto de esa división es 0, por lo que módulo de 10 y 5 es igual a 0.

Sin embargo, si se divide 9 y 5, la división no es exacta, el resultado es 1 y el resto 4, por lo que módulo de 9 y 5 es igual a 4.

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
```

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
```

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 (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja, como se verá en el siguiente capítulo de programación avanzada. 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
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. 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
```

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. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (a es menor que b, b es menor que c, A es menor que a, etc.)

Ejemplos de uso de operadores

En la Unidad 1 enumeramos los distintos tipos de operadores. Ahora vamos a ver un par de ejemplos de éstos para que nos ayuden a hacernos una idea más exacta de lo que son.

En el primer ejemplo vamos a realizar una suma utilizando el operador suma.

```
3 + 5
```

Esta es una expresión muy básica que no tiene mucho sentido ella sola.

Hace la suma entre los dos operandos número 3 y 5, pero no sirve de mucho porque no se hace nada con el resultado.

Normalmente se combinan más de un operador para crear expresiones más útiles. La expresión siguiente es una combinación entre dos operadores, uno realiza una operación matemática y el otro sirve para guardar el resultado.

```
miVariable = 23 * 5
```

En el ejemplo anterior, el operador * se utiliza para realizar una multiplicación y el operador = se utiliza para asignar el resultado en una variable, de modo que guardemos el valor para su posterior uso. Los operadores se pueden clasificar según el tipo de acciones que realizan.

Diferencias entre == y ===

En JavaScript existen dos operadores para realizar comparación de igualdad, estos son el operador de igualdad **==** y el operador de identidad o igualdad estricta **===**.

En casos comunes, como comparar si 1 es igual a 1, ambos operadores devuelven el mismo resultado, sin embargo en muchas otras ocasiones van a devolver resultados opuestos.

Es por esto que entender la diferencia en su funcionamiento es importante para un programador en JavaScript.

Operador ==

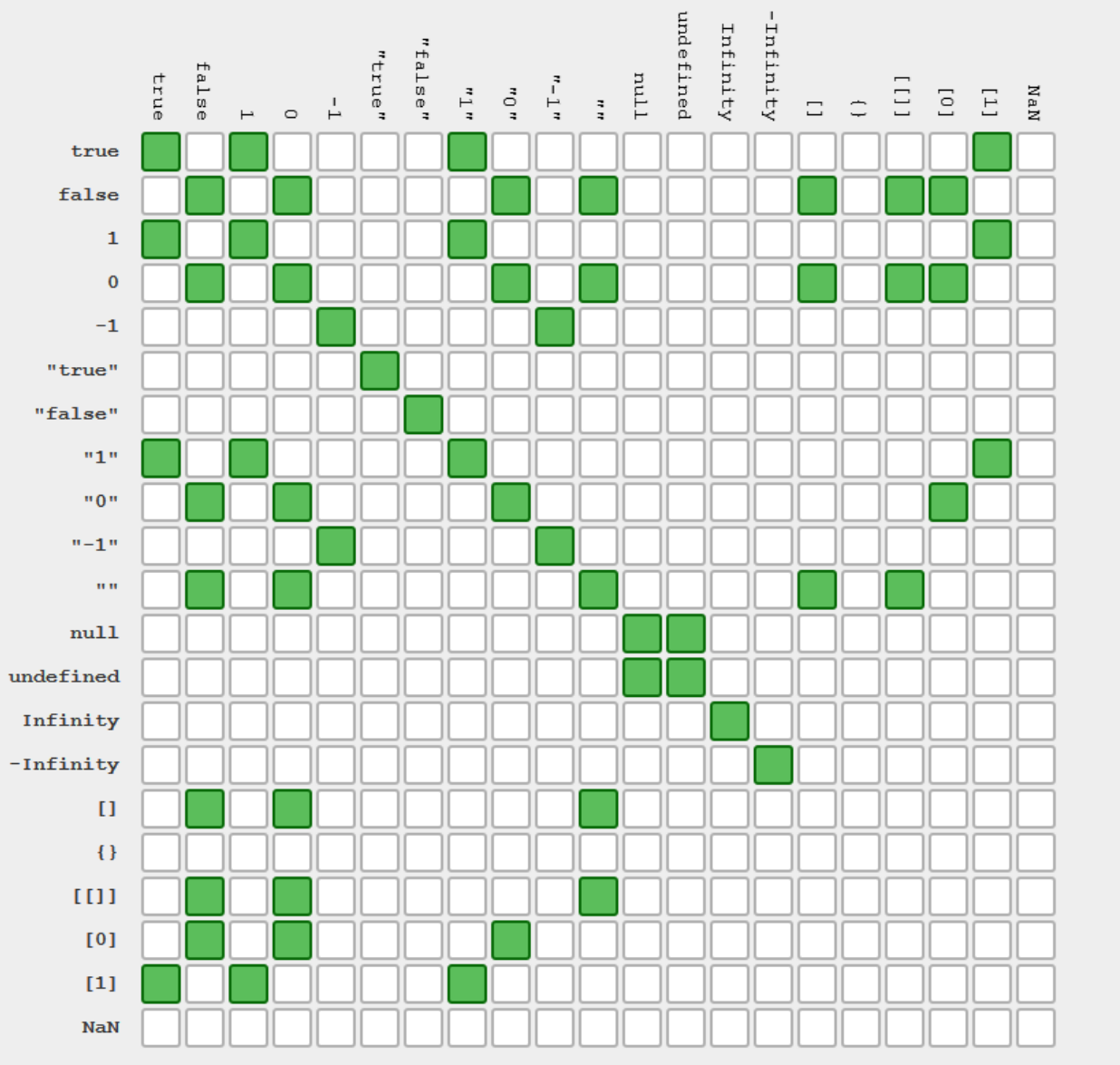
El operador de igualdad == convierte los operandos si no son del mismo tipo, después aplica comparación de igualdad estricta.

```
1   ==  1     // true

"1"  ==  1     // true

 1   == '1'    // true

 0   == false  // true
```

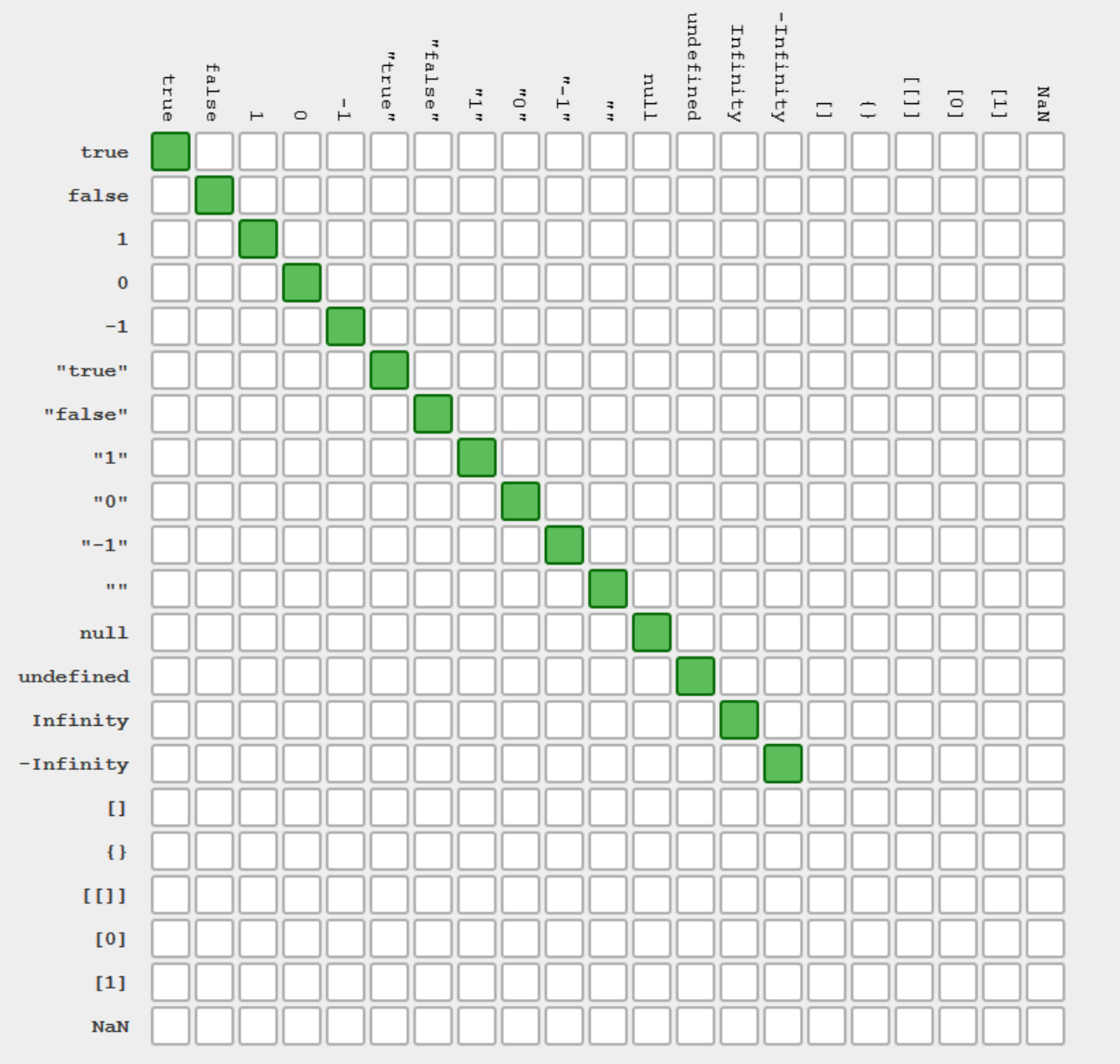


Operador ===

El operador de identidad === devuelve true si los operandos se consideran igual sin aplicar ningún tipo de conversión.

```
3 === 3   // true

3 === '3' // false
```


true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
NaN

Lo que vimos:

En esta unidad trabajamos con Variables y operadores

Lo que viene:

Comenzaremos a trabajar vectores y ciclos.