

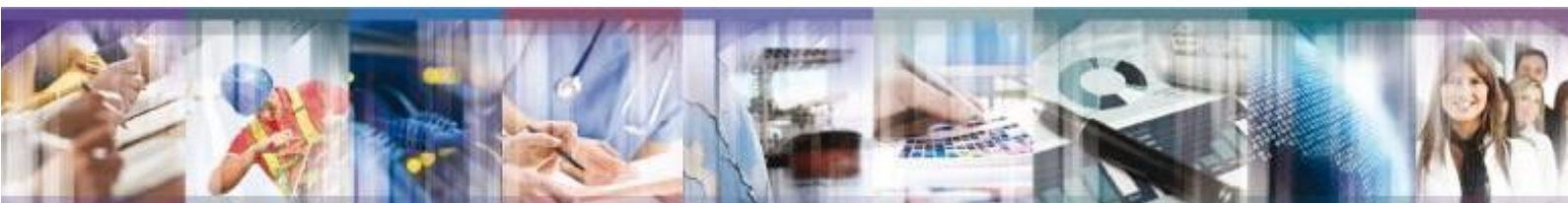


**UTN.BA** FACULTAD  
REGIONAL  
BUENOS AIRES  
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

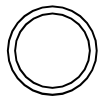
**Centro de  
e-Learning**

# Desarrollo Web

## con JavaScript



[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)



## Temario:

### Temario:

1. Variables
2. Alcance de las variables
3. Adquiriendo algunas buenas prácticas.
4. Operadores
5. Hablemos de condicionales
6. Estructura if
7. Estructura if/else
8. Estructura switch
9. Introduciendo a los objetos
10. Bibliografía utilizada

# 1. Variables

En JavaScript, definimos variables a elementos que nos sirven para guardar un valor en la computadora. Es un espacio en la memoria que le podemos asignar un nombre y que nos va a servir para poder guardar valores e ir accediendo a ellos en el momento que lo necesitemos.

Pero... ¿qué es un valor en JavaScript? Un valor es el componente más simple de JavaScript que puede ser desde un número o una letra, hasta una función y mucho más, hay muchos tipos de valores (ya en la unidad anterior vimos algunos como Strings, Numbers y Booleans), pero no nos profundizaremos ahora en ello ya que vamos a ir viéndolos unidad a unidad.

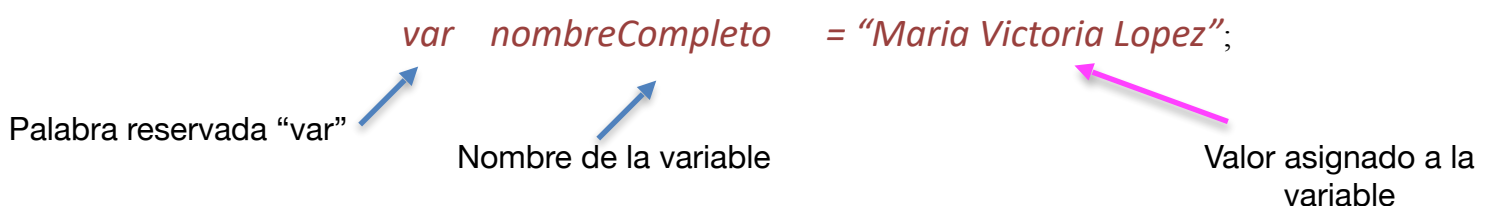
Entonces, teniendo en cuenta la definición de variable, podríamos pensarlo como un buzón donde vamos a poner cosas como una sentencia, y después le damos a la variable una dirección (en este caso un nombre) que puede usar para ir a buscar esa sentencia más tarde.

Estos nombres no deben contener espacios y pueden utilizar tanto letras mayúsculas como minúsculas; y al igual que en un buzón, nosotros podemos cambiar el contenido de esa variable, darle otros valores a lo largo del programa en caso que necesitemos.

## Sintaxis

Para poder decirle a JavaScript que lo que estamos definiendo es una variable, utilizamos la palabra reservada **var** seguido por el nombre que le vamos a dar a esa variable, luego un signo igual (=) seguido por el valor asignado a esa variable; de esta forma le estamos diciendo al navegador que guarde en esa variable todos los valores que se encuentren del lado derecho del igual, de esta forma cada vez que en el código escribamos ese nombre definido, vamos a estar invocando al valor o valores guardados en dichas variables.

*var nombreCompleto = "Maria Victoria Lopez";*



Palabra reservada "var"

Nombre de la variable

Valor asignado a la variable

Viendo este ejemplo podemos notar que cada vez que llamemos a nombreCompleto este me va a estar trayendo el string “Maria Victoria Lopez”; en caso que en vez de definir ese valor hubiera colocado el valor 13435223, al invocar a dicha variable me hubiera traído el número que le definimos.

Otro ejemplos serian:

```
var edad = 35 ;  
var verdadero = true ;  
var dato_inicial = “ “;
```

## 2.Funciones

Como vimos en el punto anterior podemos ahora guardar valores en variables para utilizarlas cuando necesitemos. Ahora tendríamos que ver como es que vamos a utilizarlo, como le decimos al navegador o al programa que tiene que hacer.

Para ello vamos a utilizar funciones, un tipo de valor que nos permite indicarle al programa las acciones que necesitamos que haga sobre esas variables que definimos previamente o utilizando dichas variables .

Una función no es más que bloques de código que nos permiten realizar tareas , son las instrucciones que le damos a los valores o sobre esos valores.

La finalidad de una función no es más que ejecutar acciones y devolver un resultado o respuesta.

Al ser un valor podemos incluso guardarla en una variable e invocarla las veces que necesitemos ejecutarla; para evitarnos repetir todo el instructivo de acciones de esa función.

A las funciones podemos tanto crearlas nosotras como utilizar algunas que tiene JavaScript ya previamente armadas, lo que nos alivia un poco el trabajo; a estas últimas las llamamos ***funciones nativas*** .

Para crear nosotros nuestras propias funciones debemos primero definir las siguiendo alguna de las siguientes estructuras:

Una vez ya definida la función , podemos invocar las mismas, las veces que queramos para

## Como se definen ??

nombre que queremos darle

valor/es de entrada

Palabra reservada  
"function"

```
function name(params){  
    //desarrollo de la funcion  
    return // palabra reservada para devolver lo que queremos  
};
```

valor/es de entrada

nombre que queremos darle

// Otra forma de escribirlo

```
var name = (params) => {  
    //desarrollo de la funcion  
    return // palabra reservada para devolver lo que queremos  
};
```

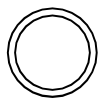
## Como se invocan ??

name(UnValor)

Nombre de la funcion definida anteriormente

Un parametro o valor que queremos que procese

En cuanto a las funciones nativas que nos provee JavaScript, contamos con una amplia variedad de funciones, de las cuales vamos a nombrar alguna de ellas, que suelen ser las más utilizadas:



**console.log()** →

Nos permite ver por consola los parametros que le indiquemos.

**alert()** →

Nos permite ver en pantalla un mensaje de alerta.

**prompt()** →

Permite que el usuario ingrese informacion .

**parseInt()** →

Transforma a numero **entero** el parametro ingresado

**parseFloat()** →

Transforma a numero **decimal** el parametro ingresado

**typeof()** →

Nos indica que tipo de dato es el parametro ingresado  
(string-boolean-number)

**toString()** →

Transforma a string el parametro ingresado

Algo a tener en cuenta al momento de usar o crear funciones es que podemos combinarlas según vamos necesitando; eso quiere decir que en caso que necesitemos, por ejemplo, que se ejecuten varias acciones de las cuales algunas las hace una función nativa, podemos utilizarla dentro de nuestra función creada.

Ejemplo:

```
/* Definimos una variable */  
  
var nombre = "Juan";  
  
/* Definimos una funcion */  
function mostrarNombre (dato) {  
    console.log(dato)  
};  
  
/* Ejecutamos la accion */  
  
mostrarNombre(nombre) // ----> esta funcion va a hacer que veamos por consola el valor "Juan"
```

### 3. Alcance de las variables

Ahora que ya logramos poder entender qué es una función y como armarla o utilizarla, podemos profundizar un poco sobre el alcance de las variables, o como se dice en el ámbito del desarrollo **scope**.

El alcance de una variable no es más que el ámbito en donde están disponibles las variables, es decir que nosotros podemos hacer que las variables puedan utilizarse en cualquier lugar de mi código JavaScript o que solamente exista en algunos lugares. De esta forma podemos limitar el espacio de memoria que vamos a estar utilizando, es decir que tal vez, necesitemos utilizar un valor para una acción en particular únicamente y no nos interese seguir guardándose luego, por lo que si limitamos esa variable a solamente existir para esa acción únicamente, dejamos más espacio de memoria libre para variables que tal vez necesitemos usar en varias acciones.

Este tipo de variables que solamente “existen” en algunas acciones en particulares, se dice que son **variables locales**.

Estas **variables locales** se definen únicamente dentro de las funciones que las necesitan, por lo que solamente van a existir cuando se ejecute dicha función.

En cambio, aquellas variables que necesitamos utilizar en varias acciones distintas, por lo que necesitamos que existan más allá de cualquier función, se llaman **variables globales** y se definen al inicio del archivo JavaScript por fuera de cualquier función.

Esto permite que yo pueda invocarlas dentro de cualquier función y accionar según necesite, mientras que en las variables locales, solamente voy a poder invocarlas dentro de la misma función en la que se encuentran definidas.

## 4. Adquiriendo algunas buenas prácticas

Con todo este nuevo conocimiento adquirido en la unidad hasta ahora, podemos aprovechar para marcar algunas buenas prácticas sobre las variables en JavaScript:

- Al momento de nombrar las variables estas solamente pueden comenzar con una letra( la cual puede ser mayúscula o minúscula ) , el símbolo pesos ( \$ ) o un guión bajo ( \_ ).
- No podemos utilizar como nombre de variables alguna de las palabras reservadas ni tampoco , en caso que utilizar un nombre que se compone de varias palabras, debe contener espacios.
- Tener en cuenta al momento de nombrar e invocar variables el uso de mayúsculas y minúsculas, ya que JavaScript es case-sensitive( es decir que es susceptibles a mayúsculas y minúsculas, por lo que no va a tomar como misma variable a una que tiene el mismo nombre en mayúscula que en minúscula).
- En JavaScript se suele agregar comentarios para ayudar al orden del código. Para ello contamos con dos maneras de agregar comentarios:

➡ `//` : Que nos permite agregar comentarios de una línea.

### Ejemplo:

`// Este es un comentario en línea`

➡ `/**` : Que nos permite agregar comentarios de más de una línea. Para utilizar este último lo que debemos hacer es encerrar el comentario entre estos signos.

### Ejemplo:

`/* Este es un comentario que me permite`

`hacer un comentario de varias líneas. */`



- Al finalizar cada instrucción, se recomienda colocar un punto y coma ( ; ) y generar un salto de línea para que el código quede legible, prolijo y asegurarnos de no producir errores en la sintaxis.

### Ejemplo:

```
var precio = 1200 ;
```

```
var apellido = "Perez";
```

```
console.log(precio);
```

## 5. Operadores

Para poder realizar alguna de las acciones, los programas que desarrollamos necesitan de operadores.

Estos nos van a permitir tomar valores y darle sentido a las instrucciones e incluso poder tanto obtener nuevos valores como nueva información sobre los valores obtenidos.

Teniendo en cuenta que existen una amplia variedad de operadores, en esta sección vamos a ocuparnos de los tres grupos principales de operadores:

### Operadores Aritméticos:

Son aquellos que ya conocemos que nos permiten usar números y realizar operaciones con ellos.

Estos utilizan los siguientes símbolos :

**+** (suma)

**-** (resta)

**\*** (multiplicación)

**/** (division)

## Operadores De Comparación:

Son aquellos que producen una respuesta afirmativa(true) o negativa(false) a una pregunta de comparación. Es utilizada para comparar dos valores.

Estos operadores son:

Igual a (==)	Solo compara la igualdad del valor en sí de cada uno.
Igual a (===)	Compara igualdad entre el valor y tipo de valor.
Distinto a (!=)	Solo compara la desigualdad del valor en sí.
Distinto a (!==)	Compara la desigualdad del valor y tipo.
Mayor a (>)	Compara cuál de los dos valores es mayor
Menor a (<)	Compara cuál de los dos valores es menor
Mayor o igual a (>=)	Compara si alguno de los valores es mayor o igual que el otro
Menor o igual a (<=)	Compara si alguno de los valores es menor o igual que el otro

## Operadores Lógicos:

Son aquellos que toman dos operandos y los evalúan, dando como resultado una respuesta afirmativa (true),o negativa (false); al igual que los operadores de comparación.

Ellos son:

### **AND (&&):**

Nos permite preguntarnos si dos cosas son simultáneamente verdaderas (es decir que son **true**).

### **OR (||):**

Nos permite preguntarnos si al menos alguna de las dos cosas son verdaderas (es decir que son **true**)..

### **NOT(!):**

Nos permite preguntarnos si algo no es lo que esperábamos que fuera(es decir si alguna de ellas es **false**) .

Tanto los operadores Lógicos como los de Comparación nos pueden servir para generar preguntas y tomar acciones según su respuesta, es decir utilizarlo como condiciones.

Esto nos va a permitir tomar distintos caminos según dicha respuesta y empezar a generar un comportamiento en el navegador dependiendo de una circunstancia en particular.

Por ahora , en una primera instancia vamos a armar estas condiciones con la siguiente sintaxis para que JavaScript lo pueda entender.

Esta consta de la comparación utilizando el operador , seguido de un signo de pregunta donde lo que coloquemos luego de ella se referirá a la acción a tomar en caso que la devolución de la operación de verdadero ,luego se colocan dos puntos ( : ) y seguido a esto, la acción que se tomara en caso que la operación dé como resultado falso.

**comparación con los operadores ? acción si el operador devuelve *true* : acción si devuelve *false***

### Ejemplo

```
var numero1 = 10 ;  
var numero2 = 20 ;  
  
numero1 > 100 ? console.log("es mayor a 100") : console.log("es menor a 100");
```

## Hablemos de condicionales

A lo largo de este curso pudimos aprender a darle comandos al navegador para que realice las acciones que necesitamos en nuestro sitio. Sin embargo, hay situaciones en donde nuestro sitio no debe realizar un procedimiento unilateral, sino que el proceder va a depender de las decisiones o acciones que tome el usuario. Por eso, nos encontramos con la necesidad de establecer condicionales para poder establecer qué acción realizará el sitio.

Estos condicionales son estructuras que nos provee JavaScript y que nos ayudan a tener un código claro, con la implementación de buenas prácticas y un abanico de acciones a realizar dentro de dicha estructura.

Teniendo en cuenta esto último, podemos decir que los condicionales pueden ser simples o anidados. Si hablamos de condicionales simples, nos referimos a estructuras que evalúan la condición y finalizan con una acción para cada resultado de la misma. En cambio, un condicional anidado, como su nombre lo indica, evalúa la condición y en alguna de las acciones a tomar crea un nuevo condicional generando una sucesión de acciones en cadena generadas por la evaluación de condicionales.

En una unidad anterior introdujimos un tipo de estructura condicional llamada **operador ternario o condicional**, que constaba de evaluar el resultado de un operador y, según su resultado fuera **true** o **false**, accionaba de una manera o de otra.

### SINTAXIS

*condición ? accionEnCasoDeSerLaRespuestaTrue : accionEnCasoDeSerLaRespuestaFalse*

### EJEMPLO

*llueve === true ? alert("podemos ir al parque") : alert("mejor quedémonos en casa")*

Este tipo de condicional se suele utilizar en acciones cortas que no resuelven en muchas acciones ya que deben escribirse en una única línea; en caso de anidar, resulta poco visual y un poco engorroso. Por eso, también contamos con otro tipo de estructuras condicionales, las cuales veremos a continuación.

## .Estructura if

Esta estructura se compone de la palabra reservada **if** seguido por unos paréntesis donde se coloca la condición a cumplir. Luego, van unas llaves donde se escribe las acciones a ejecutar, o en caso de ser un condicional anidado, la siguiente estructura **if**.

### SINTAXIS

```
if ( condición ) {  
    Ejecuto un código  
}
```

Si la condición es **true**, ejecuto el código entre llaves.

La estructura **if** nos permite indicar que, en caso de que la condición que se evalúe resulte verdadera (es decir que dé como resultado **true** ) se realizarán las acciones que se indican dentro de las llaves. Una vez ejecutada las acciones dentro del if, continúa ejecutando el código que se redacte luego de esta estructura.

El condicional **if** se puede leer como: “solo si se cumple esta condición, ejecutar las acciones dentro”.

Tenemos que tener en cuenta que este tipo de estructuras nos permite agrupar un conjunto de acciones y proceder de distintas formas, relacionando el accionar del usuario y provocando una mejor interacción entre el sitio y el usuario.

La estructura if solamente se fija en lo que va a ejecutar si se cumple esa condición. Entonces.. ¿qué pasa cuando necesitamos indicarle al programa que ejecute otro conjunto de acciones en caso que no cumpla con la condición del if?

## Estructura if/else

Cuando nos encontramos en una situación en la cual necesitamos manipular los dos caminos posibles que puede tomar un condicional (es decir, si la condición da como resultado true o si resuelve en false), podemos utilizar la estructura if/else.

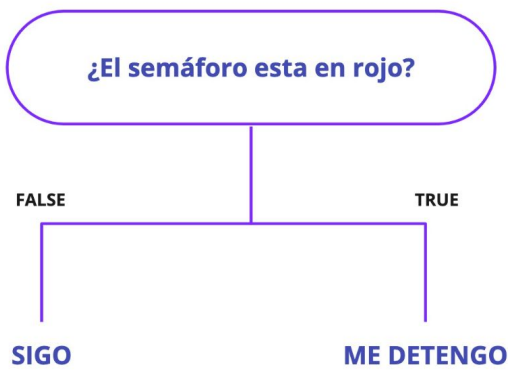
Esta estructura nos va a indicar el grupo de acciones a realizar tanto si el resultado de la condición es **true** como cuando sea **false**.

Lo que ocurrirá con esta estructura es que, dependiendo de la condición, tomará alguno de los dos caminos y; Una vez finalizada la ejecución de las acciones indicadas, continuará con la ejecución del código programadas por fuera de la estructura.

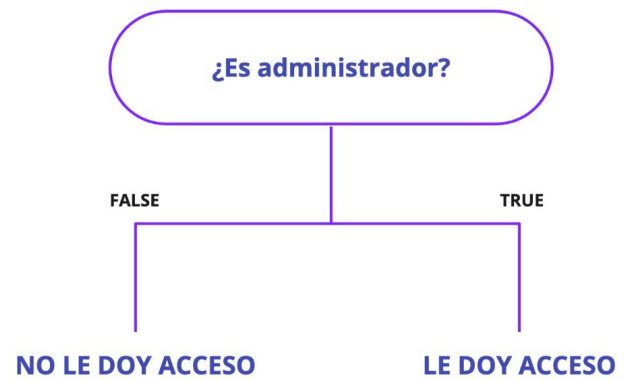
### SINTAXIS

```
if ( condición ) {  
    Ejecuto si se cumple la condición  
} else {  
    Ejecuto si no se cumple la condición  
}
```

### EJEMPLO



```
if (semaforo == rojo) {  
    // me detengo  
  
    else {  
        //sigo  
    }  
}
```



```
if (administrador) {  
    // le doy acceso  
  
    else {  
        //no le doy acceso  
    }  
}
```

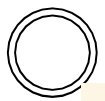
### 3. Estructura switch

Además, la estructura switch nos permite evaluar una expresión y tomar distintos caminos de acción según el resultado de esa evaluación.

Switch es una de las estructuras recomendadas cuando tenemos varias posibles respuestas para una misma condición y, a diferencia de las estructuras if o if/else, ahora vamos a poder evaluar resultados por fuera del **true** y **false**.

Esta estructura se compone por las palabras reservadas **switch**, **case**, **break** y **default**. Se escribe de la siguiente forma.

#### SINTAXIS



```
switch (expresión) {  
    case valor1:  
        Sentencias a ejecutar si la expresión tiene como valor a valor1  
        break  
    case valor2:  
        Sentencias a ejecutar si la expresión tiene como valor a valor2  
        break  
    case valor3:  
        Sentencias a ejecutar si la expresión tiene como valor a valor3  
        break  
    default:  
        Sentencias a ejecutar si el valor no es ninguno de los anteriores  
}
```

Imagen extraída de <https://desarrolloweb.com/>

Teniendo en cuenta la imagen anterior, el **switch** evalúa la expresión que se encuentra entre los **paréntesis** y, dependiendo del resultado, irá por las acciones que se encuentren dentro del **case** que coincida con ese resultado. Si la respuesta que recibe no está contemplada en ninguno de los **case**, procederá a ejecutar las declaraciones hechas dentro de la opción **default**.

En cuanto a la palabra **break**, indica que una vez finalizada la ejecución de las acciones definidas en el **case** o en el **default** (según el camino que resuelva el switch) va a continuar con la ejecución del código programado por fuera del switch.

## EJEMPLO

```
switch (colorSemaforo) {  
    case "Amarillo":  
        //precaución proximo a cambio  
        break;  
    case "Rojo":  
        //detenerse  
        break;  
    case "Verde":  
        //avanzar  
        break;  
    default:  
        //no pertenece a unas de las opciones  
        del semaforo  
}
```

En este ejemplo lo que podemos observar es que, dependiendo el color del semáforo la acción a realizar.

## 4. Introducción a los objetos

En anteriores unidades aprendimos sobre tipo de datos y cómo almacenarlos en variables. También pudimos ver que existen los arrays que nos permiten agrupar en forma de lista a varios datos que consideremos.

Pero va a ver situaciones en la que necesitemos acceder a características de esos valores o datos que estamos almacenando. Para ello, nos ayudarán los objetos.

Los objetos son estructuras que guardamos en variables que nos permiten almacenar múltiples propiedades de datos. Para entenderlo mejor, usemos un objeto de la vida real: una taza. Viendo una taza puedo definir distintas propiedades: tamaño, color, peso, el material con el que está hecho, etc. Estas propiedades modifican su valor según, en este caso, la taza que esté observando. Es decir que una taza puede ser de color rojo, de cerámica y para café; o puedo tener otra taza que sea de vidrio, más grande, de forma mas ovalada... pero ambas son tazas.

Con esto lo que queremos decir es que los objetos están compuestos de propiedades y distintos valores para dichas propiedades, según el objeto que estemos definiendo.

JavaScript lo representa con llaves ({} ) donde dentro coloca el tipo de propiedad (denominada claves o keys ) seguido por dos puntos ( : ) y luego el valor de esa propiedad (denominado value o valor). Cada clave o key con su valor correspondiente se separa con un punto y coma ( ; ) exceptuando la última propiedad.

En la siguiente unidad estaremos viendo cómo poder manipularlo y acceder a los valores.

### SINTAXIS

```
var/let/const nombreDelObjeto ={  
  
    nombreDeLaClave : suValor ;  
    nombreDeOtraClave : suValor  
}
```





```
var/let/const nombreDelObjeto ={  
    nombreDeLaClave : suValor ;  
    nombreDeOtraClave : suValor  
}
```

## EJEMPLO

## 6. Bibliografía utilizada y sugerida

<https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/switch>

<https://es.javascript.info/object>

<https://www.javascript.com/learn/objects>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/if...else>

[https://www.w3schools.com/js/js\\_if\\_else.asp](https://www.w3schools.com/js/js_if_else.asp)

<https://desarrolloweb.com/articulos/546.php>

<https://jsparagatos.com/>

<https://developer.mozilla.org/es/docs/Glossary/Variable>

<https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/var>

[https://www.w3schools.com/js/js\\_variables.asp](https://www.w3schools.com/js/js_variables.asp)

[https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions\\_and\\_Operators#comparacion](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators#comparacion)

[https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions\\_and\\_Operators#aritmeticos](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators#aritmeticos)

[https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions\\_and\\_Operators#logico](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators#logico)

