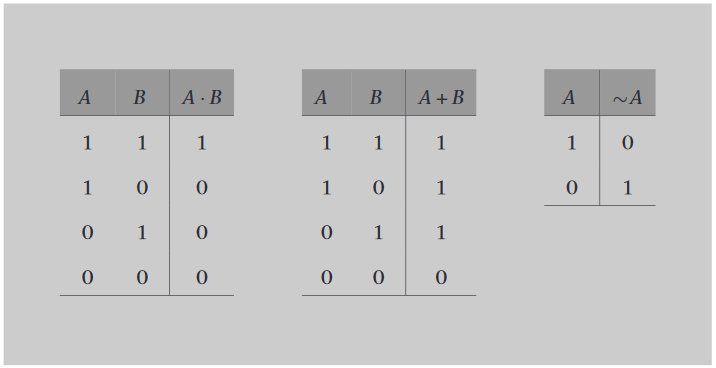
**Álgebra Booleana**

A continuación se especifican las tablas de valores para los tres operadores básicos. En las columnas de las expresiones A y B aparecen todos los posibles pares de valores que pueden tomar. En la columna de la expresión compleja formada con el operador se recoge el valor que corresponde al par de valores de las expresiones más simples que hay en cada fila



**Interpretación de las tablas de verdad**

Para entender cómo se debe interpretar la tabla de verdad de un operador tomemos, por ejemplo, la tabla del operador conjunción: cuando *A* = 1 y *B* = 1 el valor de la expresión compleja *A*·*B* = 1 (es la primera fila de la tabla); respectivamente, cuando *A* = 1 y *B* = 0 el valor de la expresión compleja *A*·*B* = 0 (es la segunda fila de la tabla); y así sucesivamente.

Observad que las tablas se corresponden con los valores de la expresión dada en la tabla de presentación de los operadores en el subapartado anterior:

* Producto lógico o conjunción. La expresión *A*·*B* toma el valor 1 si las expresiones *A* y *B* lo toman simultáneamente y toma el valor 0 en cualquier otro caso.
* Suma lógica o disyunción. La expresión *A* + *B* toma el valor 1 si al menos una de las expresiones *A* o *B* toman el valor 1 y toma el valor 0 en cualquier otro caso.
* Complementación o negación. La expresión \_ *A* toma el valor 1 si la expresión *A* toma el valor 0, y toma el valor 0 si la expresión *A* toma el valor 1.

**Tipos booleanos**

En el ejemplo que queremos resolver, la condición que queremos ver si se cumple o no es que x sea mayor que cero. Python provee las llamadas \*expresiones de comparación\* que sirven para comparar valores entre sí, y que por lo tanto permiten codificar ese tipo de pregunta. En particular la

De esta forma, 5 > 3 es una expresión booleana cuyo valor es True, y 5 < 3 también es una expresión booleana, pero su valor es False

>>> 5 > 3

True

>>> 3 > 5

False >>>

Las expresiones booleanas de comparación que provee Python son las siguientes:

|  |  |
| --- | --- |
| **Expresión** | **Significado** |
| a == b | a es igual a b |
| a != b | a es distinto de b |
| a < b | a es menor que b |
| a <= b | a es menor o igual que  b |
| a > b | a es mayor que b |
| a >= b | a es mayor o igual que b |

A continuación, algunos ejemplos de uso de estos operadores:

>>> 6 == 6

True

>>> 6 != 6

False

>>> 6 > 6

False

>>> 6 >= 6

True

>>> 6 > 4

True

>>> 6 < 4

False

>>> 6 <= 4

False

>>> 4 < 6

True

## Operadores lógicos

De la misma manera que se puede operar entre números mediante las operaciones de suma, resta, etc., también existen tres operadores lógicos para combinar expresiones booleanas: and (y), or (o) y not (no).

El significado de estos operadores es igual al del castellano, pero vale la pena recordarlo:

|  |  |
| --- | --- |
| **Expresión** | **Significado** |
| a and b | El resultado es True solamente si a es True y b es True de lo contrario el resultado es Fals  e |
| a or b | El resultado es True si a es True o b es True de lo contrario el resultado es False |
| not a | El resultado es True si a es False de lo contrario el resultado es False |

a > b and a > c es verdadero si a es simultáneamente mayor que b y que c.

>>> 5 > 2 and 5 > 3

True

>>> 5 > 2 and 5 > 6

False

a > b or a > c es verdadero si a es mayor que b o a es mayor que c.

>>> 5 > 2 or 5 > 3

True

>>> 5 > 2 or 5 > 6

True

>>> 5 > 8 or 5 > 6

False

not (a > b) es verdadero si a > b es falso (o sea si a <= b es verdadero).

>>> 5 > 8

False

>>> not (5 > 8)

True

>>> 5 > 2

True

>>> not (5 > 2)

False

**Tipos booleanos: True y False**

## Tipo booleanos

Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como verá más adelante.

|  |  |  |  |
| --- | --- | --- | --- |
| Clase | Tipo | Notas | Ejemplo |
| bool | Números | Valor booleano falso. | False |
| bool | Números | Valor booleano verdadero. | True |

Una variable booleana es una variable que sólo puede tomar dos posibles valores: True (verdadero) o False (falso).

En Python cualquier variable (en general, cualquier objeto) puede considerarse como una variable booleana. En general los elementos nulos o vacíos se consideran False y el resto se consideran True.

Para comprobar si un elemento se considera True o False, se puede convertir a su valor booleano mediante la función bool().

>>> bool(0)

False

>>> bool(0.0)

False

>>> bool("")

False

>>> bool(None)

False

>>> bool(())

False

>>> bool([])

False

>>> bool({})

False

>>> bool(25)

True

>>> bool(-9.5)

True

>>> bool("abc")

True

>>> bool((1, 2, 3))

True

>>> bool([27, "octubre", 1997])

True

>>> bool({27, "octubre", 1997})

True

# Operadores lógicos

Los **operadores lógicos** son unas operaciones que trabajan con valores booleanos.

* and: "y" lógico. Este operador da como resultado True si y sólo si sus dos operandos son True:

>>> True and True

True

>>> True and False

False

>>> False and True

False

>>> False and False

False

* or: "o" lógico. Este operador da como resultado True si algún operando es True:

>>> True or True

True

>>> True or False

True

>>> False or True

True

>>> False or False

False

**Nota**: En el lenguaje cotidiano, el "o" se utiliza a menudo en situaciones en las que sólo puede darse una de las dos alternativas. Por ejemplo, en un menú de restaurante se puede elegir "postre o café", pero no las dos cosas (salvo que se pague aparte, claro). En lógica, ese tipo de "o" se denomina "o exclusivo" (xor).

* not: negación. Este operador da como resultado True si y sólo si su argumento es False:

>>> not True

False

>>> not False

True

# Expresiones compuestas

Si no se está acostumbrado a evaluar expresiones lógicas compuestas, se recomienda utilizar paréntesis para asegurar el orden de las operaciones.

Al componer expresiones más complejas hay que tener en cuenta que Python evalúa primero los not, luego los and y por último los or, como puede comprobarse en los ejemplos siguientes:

* El operador not se evalúa antes que el operador and:

>>> not True and False

False

>>> (not True) and False

False

>>> not (True and False)

True

* El operador not se evalúa antes que el operador or:

>>> not False or True

True

>>> (not False) or True

True

>>> not (False or True)

False

* El operador and se evalúa antes que el operador or:

>>> False and True or True

True

>>> (False and True) or True

True

>>> False and (True or True)

False

>>> True or True and False

True

>>> (True or True) and False

False

>>> True or (True and False)

True

* Si en las expresiones lógicas se utilizan valores distintos de True o False, Python utiliza esos valores en vez de True o False.

>>> 3 or 4

3

* Si se quieren mostrar valores booleanos, se puede convertir el resultado a un valor booleano:

>>> 3 or 4

3

>>> bool(3 or 4) # Verdadero porque 3 es diferente de 0

True

**Nota**: No suele ser muy habitual tener que utilizar este tipo de expresiones. En la lecció[n](https://www.mclibre.org/consultar/python/lecciones/python-detalles.html#expresiones-logicas) [Detalles de Python](https://www.mclibre.org/consultar/python/lecciones/python-detalles.html#expresiones-logicas) se comenta con más detalle cómo Python elige la respuesta a estas expresiones.

# Comparaciones

Las comparaciones también dan como resultado valores booleanos:

* >Mayor que; **<** Menor que;

>>> 3 > 2

True

>>> 3 < 2

False

* **>=** Mayor o igual que; **<=** Menor o igual que;

>>> 2 >= 1 + 1

True

>>> 4 - 2 <= 1

False

* **==** Igual que; **!=** Distinto de;

>>> 2 == 1 + 1

True

>>> 6 / 2 != 3

False

Es importante señalar que en matemáticas el signo igual se utiliza tanto en las asignaciones como en las comparaciones, mientras que en Python (y en otros muchos lenguajes de programación):

* un signo igual (=) significa asignación, es decir, almacenar un valor en una variable
* mientras que dos signos iguales seguidos (==) significa comparación, es decir, decir si es verdad o mentira que dos expresiones son iguales

Cuando se aprende a programar es habitual confundir una cosa con la otra (el error más frecuente es escribir una sola igualdad en las comparaciones), por lo que se recomienda prestar atención a este detalle.

Python permite encadenar varias comparaciones y el resultado será verdadero si y sólo si todas las comparaciones lo son.

>>> 4 == 3 + 1 > 2

True

>>> 2 != 1 + 1 > 0

False

Encadenar comparaciones no está permitido en otros lenguajes como PHP, en los que las comparaciones deben combinarse mediante operadores lógicos and, lo que también puede hacerse en Python:

>>> 4 == 3 + 1 and 3 + 1 > 2

True

>>> 2 != 1 + 1 and 1 + 1 > 0

False