



---

# INTRODUCCIÓN A LA PROGRAMACIÓN CON PYTHON

---



## UNIDAD II Tipos de datos

MSc Ing Emiliano López

# Introducción a la Programación con Python

**Autor:** Emiliano López - [elopez@fich.unl.edu.ar](mailto:elopez@fich.unl.edu.ar)

**Fecha:** 13/10/2016 16:10 - [última versión disponible]

## Unidad II: Tipos de datos

<b>1</b>	<b>Introducción</b>	<b>4</b>
<b>2</b>	<b>Tipos de datos Numéricos</b>	<b>4</b>
2.1	Enteros	4
2.2	Reales	4
2.3	Complejos	5
<b>3</b>	<b>Cadenas de caracteres</b>	<b>5</b>
<b>4</b>	<b>Lógicos</b>	<b>5</b>
<b>5</b>	<b>Operadores</b>	<b>6</b>
5.1	Operadores relationales	6
5.1.1	Entre cadenas de caracteres	7
5.1.2	Entre números	7
5.1.3	Entre valores lógicos	8
5.2	Operadores lógicos	8

## LICENCIA CC BY-SA 4.0



*Introducción a la Programación con Python* por Emiliano López se distribuye bajo una **Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional**.

A continuación una traducción de la licencia que podría diferir de la [original](#):

### Usted es libre para:

- Compartir - copiar y redistribuir el material en cualquier medio o formato
- Adaptar - remezclar, transformar y crear a partir del material

Para cualquier propósito, incluso comercialmente

El licenciatario no puede revocar estas libertades en tanto usted siga los términos de la licencia

### Bajo los siguientes términos:

- Atribución - Usted debe darle crédito a esta obra de manera adecuada (ver \*), proporcionando un enlace a la licencia, e indicando si se han realizado cambios (ver \*\*). Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo del licenciatario.
- Compartir Igual - Si usted mezcla, transforma o crea nuevo material a partir de esta obra, usted podrá distribuir su contribución siempre que utilice la misma licencia que la obra original.

\* Si se suministran, usted debe dar el nombre del creador y de las partes atribuidas, un aviso de derechos de autor, una nota de licencia, un aviso legal, y un enlace al material. Las licencias CC anteriores a la versión 4.0 requieren que usted provea el título del material si se incluye, y pueden tener otras ligeras diferencias.

\*\* En 4.0, debe indicar si ha modificado el material y mantener una indicación de las modificaciones anteriores

## 1 Introducción

Como vimos en la Unidad previa, las variables pueden contener diferentes tipos de datos: numéricos (enteros o flotantes), cadenas de caracteres (contenido encerrados con comillas) y lógicos (*True* y *False*). Si bien hasta ahora fue transparente el tipo de dato que manejábamos, es útil conocerlo ya que cada uno tiene asociadas ciertas funcionalidades. Por ejemplo, vimos que multiplicar un número entero por una cadena de caracteres es una operación permitida y no así una suma.

A continuación veremos en mayor profundidad los tipos de datos incorporados en Python programando los ejemplos directamente sobre el intérprete mejorado ipython.

## 2 Tipos de datos Numéricos

Los números pueden ser enteros, reales (también denominados de punto flotante) o complejos.

### 2.1 Enteros

Los números enteros son aquellos que no tienen decimales (ya sean positivos o negativos además del cero). En Python se representan mediante el tipo *int* (del inglés, *integer*) que puede ser consultado a través de la función *type()*. Por ejemplo:

```
a = 4
type(a)
int
```

### 2.2 Reales

Los números reales son los que tienen decimales. En Python se expresan mediante el tipo *float* y se los denomina flotantes.

Para representar un número real en Python se escribe primero la parte entera, seguido de un punto y por último la parte decimal. Por ejemplo:

```
real = 6.2231
```

También se puede utilizar notación científica, y añadir una *e* (de exponente) para indicar un exponente en base 10. Por ejemplo:

```
real = 0.6e-3
```

Lo que sería equivalente a  $0.6 \times 10(-3) = 0.6 \times 0.001 = 0.0006$

```
real = 8.21
type(real)
float
```

## 2.3 Complejos

Los números complejos son aquellos que están formados por una parte real y otra imaginaria. La mayor parte de los lenguajes de programación carecen de este tipo, aunque sea muy utilizado por ingenieros y científicos en general.

En Python son llamados *complex*, y se representan de la siguiente forma:

```
c= 4 + 5j
type(c)
complex
```

## 3 Cadenas de caracteres

Tal como hemos visto en la unidad anterior, las cadenas (string en inglés o str) no son más que texto encerrado entre comillas simples, dobles o triples ('cadena', "cadena","Cadenas multilíneas"). Por ejemplo:

```
a = 'Si supiera que el mundo se acaba mañana, yo, hoy todavía, \
      plantaría un árbol (Martin Luther King).'
type(a)
str
```

```
b = "La simplicidad es la máxima sofisticación (Leonardo Da Vinci)."
type(b)
str
```

```
c = '''Como le gusta el baile
      al hijo de cuca
      no le gusta el trabajo
      al hijo de cuca
      el vive de la calle
      el hijo de cuca
      no le importa un comino
      al hijo de cuca  (Pocho La Pantera, ~ año 1994).'''
type(c)
str
```

## 4 Lógicos

Por último, el tipo *lógico*, comúnmente denominado booleano. Una variable de tipo *bool* sólo puede tener dos valores posibles: *True* (verdadero) o *False* (falso). Estos valores son especialmente importantes ya que provienen de operaciones relacionales o lógicas, imprescindibles en todo algoritmo computacional. Toda operación de comparación, independientemente del tipo de dato que se esté operando, arroja un resultado lógico.

Antes de adentrarnos en las operaciones que nos arrojan este tipo de valores, veamos algunos ejemplos de variables lógicas:

```
a = True  
type(a)  
bool
```

```
b = False  
type(b)  
bool
```

En el siguiente ejemplo se asigna a la variable **c** el resultado de una operación relacional ( $10 > 2$ ) que resulta verdadera (*True*), dado que 10 es mayor que 2.

```
c = 10 > 2  
print(c)  
True
```

Al tratarse de una operación lógica, el resultado siempre será de tipo *bool*.

```
type(c)  
bool
```

## 5 Operadores

### 5.1 Operadores relacionales

Como vimos en el ejemplo anterior, los valores *bool* son además el resultado de expresiones que utilizan operadores relacionales, es decir, comparaciones entre valores. Este tipo de expresiones pueden ser analizadas como una pregunta cuya respuesta consiste en una de dos posibilidades: verdadera o falsa.

Los operadores relacionales se utilizan de la siguiente manera: OperandoA Operador OperandoB

Por ejemplo:

```
10 > 4  
True
```

En este caso el operando A es 10 y el B es 4, el resultado de aplicar el operador " $>$ " a los operandos es en este caso *True* dado que 10 es mayor que 4.

La lista completa de operadores que podemos utilizar en python es:

Operador	Descripción	Ejemplo	Resultado
<code>==</code>	¿son iguales a y b?	<code>5 == 3</code>	<code>False</code>
<code>!=</code>	¿son distintos a y b?	<code>5 != 3</code>	<code>True</code>
<code>&lt;</code>	¿es a menor que b?	<code>5 &lt; 3</code>	<code>False</code>
<code>&gt;</code>	¿es a mayor que b?	<code>5 &gt; 3</code>	<code>True</code>

Además, los de mayor (`>`) y menor (`<`) se pueden combinar con el igual (`=`) para realizar lo siguiente:

<code>&lt;=</code>	¿es a menor o igual que b?	<code>5 &lt;= 5</code>	<code>True</code>
<code>&gt;=</code>	¿es a mayor o igual que b?	<code>2 &gt;= 3</code>	<code>False</code>

Veamos una serie de operaciones relacionales entre cadenas de texto, números y valores lógicos.

### 5.1.1 Entre cadenas de caracteres

```
d = "Una cosa" == "Otra cosa"
print(d)
False
```

En este caso el operador `==` se utiliza para comparar si son iguales los operandos. Esta comparación se hace carácter a carácter, por lo que al ser diferentes las cadenas, el resultado es `False`. Lo siguiente también es `False`

```
d = "Una cosa" == "una cosa"
print(d)
False
```

Solo cuando ambas cadenas son exactamente iguales, la comparación da como resultado un valor verdadero

```
d = "Una cosa" == "Una cosa"
print (d)
True
```

Veamos el tipo:

```
type(d)
bool
```

### 5.1.2 Entre números

```
resultado = 24 > 3*7
print (resultado)
True
```

### 5.1.3 Entre valores lógicos

```
resultado = False == True
print (resultado)
False
```

En Python, como en otros lenguajes, una expresión que es verdadera tiene el valor 1, y una expresión que es falsa tiene el valor 0. Es decir, *True* es equivalente a 1 y *False* a 0.

```
a = True
resultado = a == 1
print (resultado)

b = False
resultado = b == 0
print (resultado)
```

```
True
True
```

## 5.2 Operadores lógicos

Los operadores lógicos se utilizan para combinar expresiones que arrojan valores de tipo *bool*. Al igual que los operadores relacionales, el resultado de estas operaciones son *True* o *False*.

Existen 3 tipos de operadores lógicos: *and* (y), *or* (o), y *not* (no). Veamos su uso en algunos ejemplos.

Usamos *and* para combinar dos operaciones relacionales:

```
(x > 0) and (x < 10)
```

El resultado arrojado por la operación es verdadero sólo si *x* es mayor que 0 **y a la vez** es menor que 10. Ahora usemos el operador *or*:

```
(n % 2 == 0) or (n % 3 == 0)
```

es verdadero si alguna de las condiciones es verdadera, es decir, si el número es divisible por 2 **o** es divisible por 3. Podemos leer la línea anterior como el resto de dividir *n* por 2 es igual a cero **o** el resto de dividir *n* por 3 es igual a cero.

Teniendo en cuenta que el operador % da como resultado el resto de la división (denominado como operador módulo). El resto de la división es cero cuando el dividendo y el divisor son múltiplos.

Finalmente, el operador *not* niega una expresión booleana, es decir, que cambia el resultado de la operación, si era *True* será *False* y si era *False*, será *True*, de forma que:

```
not (x > y)
```

es verdadero si la expresión es falsa, o sea, si x es menor o igual que y.

En resumen tenemos los siguientes operadores lógicos

Operador	Descripción	Ejemplo	Resultado
<b>and</b>	¿se cumple a y b?	<b>True and False</b>	<b>False</b>
<b>or</b>	¿se cumple a o b?	<b>True or False</b>	<b>True</b>
<b>not</b>	No a	<b>not True</b>	<b>False</b>

Veamos algunos ejemplos

```
a = 9
b = 16
c = 6
resultado = (a < b) and (a > c)
print (resultado)
True
```

En este caso, como ambas operaciones devuelven *True*, el resultado es verdadero.

```
a = 9
b = 16
c = 6
resultado = (a < b) and (a < c)
print (resultado)
False
```

Por el contrario, si una de las condiciones devuelve *False*, el resultado será *False*.

Veamos algunos ejemplos con el operador *or*

```
a = 9
b = 16
c = 6
resultado = (a < b) or (a < c)
print(resultado)
True
```

En este caso la primer operación es verdadera y la segunda es falsa, pero como estamos utilizando el operador **\*or\***, la variable resultado tendrá como valor *True*.

Por último, veamos un ejemplo con el operador *not*

```
a = 9
b = 16
resultado = not(a > b)
print(resultado)
True
```

En este ejemplo *a* es menor que *b*, por lo que la expresión es falsa. Sin embargo al utilizarse el operador *not* estamos cambiando el resultado por su opuesto (en este caso *True*). La expresión podría leer como "no es cierto que *a* es mayor que *b*", lo cual es una expresión cierta, y por lo tanto el valor correspondiente es *True*.

Veamos un ejemplo un poco mas complejo:

```
a = 9
b = 16
resultado = (not(a > b)) and (not(b < c))
print(resultado)
True
```

Analicemos detalladamente el ejemplo:

En este caso la expresión (*a* > *b*) es falsa, al igual que (*b* < *c*), por lo que podríamos ver a lo anterior como

```
resultado = (not(False)) and (not(False))
```

Dijimos que el operador *not* cambia el resultado de una expresión lógica por su opuesto, por lo que si seguimos desarrollando esta línea tenemos:

```
resultado = (True) and (True)
```

Como ambas expresiones son verdaderas, el valor de la variable *resultado* será *True*.

Se debe tener un especial cuidado con el orden en que se utilizan los operadores. Para asegurarnos de que estamos aplicando los operadores a una expresión particular, es recomendable utilizar paréntesis.