Tecnicatura Universitaria en Software Libre

Introducción al Desarrollo de Software

Docente: Emiliano López
Tutor: Maximiliano Boscovich

Contenidos

Introducción al Desarrollo de Software

Este documento fue generado el 2015-09-15 14:03

1 Unidad 2: Tipos básicos

Como vimos en la Unidad 1, las variables pueden contener diferentes tipos de datos, y al ser distintos, son tratados de manera diferente por Python (por ejemplo no podemos sumar un número con una letra).

Hemos visto 2 de los 3 tipos básicos que utiliza python, los cuales se dividen en: * **Números** * **Cadenas de caracteres** * **Lógicos**

1.1 Números

Los números como vimos pueden ser enteros, reales (también denominados de coma flotante) ó complejos. ### Enteros Los números enteros son aquellos números positivos o negativos que no tienen decimales (además del cero). En Python se representan mediante el tipo int (de integer, entero). Por ejemplo:

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

```
int
```

1.1.1 Reales

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

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 x 10-3 = 0.6 x 0.001 = 0.0006

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

```
float
```

1.1.2 Complejos

Los números complejos son aquellos que tienen parte imaginaria. Si no conocías de su existencia, es más que probable que nunca lo vayas a necesitar, de hecho la mayor parte de los lenguajes de programación carecen de este tipo, aunque sea muy utilizado por ingenieros y científicos en general.

En el caso de que necesites utilizar números complejos, debes saber que son llamados complex en Python, y que se representan de la siguiente forma:

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

```
complex
```

1.2 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 ('cadena'), dobles ("cadena") ó triples('''Cadenas multilíneas'''). Por ejemplo:

```
a = 'El futuro mostrará los resultados y juzgará a cada uno de \
    acuerdo a sus logros (Nikola Tesla)'
type(a)
```

```
str
```

```
b = "En realidad no me preocupa que quieran robar mis ideas, \
    me preocupa que ellos no las tengan (Nikola Tesla)"
type(b)
```

```
str
```

```
c = '''Un instrumento de poco costo y no más grande que un reloj, \
    permitirá a su portador escuchar en cualquier parte, ya sea en\
    el mar o en la tierra, música, canciones o un discurso de un \
    líder político, dictado en cualquier otro sitio distante. Del\
    mismo modo, cualquier dibujo o impresión podrá ser \
    transferida de un lugar a otro (Nikola Tesla, ~ año 1891).'''
type(c)
```

```
str
```

1.3 Lógicos

Por último, nos queda el tipo básico lógico, comunmente denominado booleano. Una variable de tipo booleano sólo puede tener dos valores: True (verdadero) y False (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como veremos más adelante. Pero veamos algunos ejemplos:

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

```
bool
```

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

```
bool
```

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

```
True
```

En este último ejemplo vemos algo particular, hemos asignado a la variable **c** el resultado de una expresión lógica (10 > 2). Python en este caso opera con la misma y asigna a la variable **c** el resultado de dicha operación, la cual en este caso es verdadera (True), dado que 10 es mayor que 2. Al tratarse se una operación lógica, el resultado siempre será de tipo booleando (bool), es decir, será verdadero o será falso.

```
type(c)
```

```
bool
```

1.3.1 Operadores relacionales

Como vimos en el ejemplo anterior, los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores).

Estos operadores, siempre se utilizan de la siguiente manera:

operando_A (operador) operando_B

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 A y B en este caso es True (cierto) dado qeu 10 es mayor que 4.

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

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

Veamos otro ejemplo, ahora con cadenas de texto:

```
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 caracter a caracter, por lo que al ser diferentes las cadenas, el resultado es False. Lo siquiente también es False

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

```
False
```

Solo cuando ambas cadenas son iguales, la comparación devuelve verdadero

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

```
True
```

El tipo como hemos visto, es booleano:

```
type(d)
```

```
bool
```

También podemos comparar números, expresiones algebráicas y expresiones lógicas.

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

```
True
```

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

```
False
```

```
a = 2*8
b = 3*8
resultado = (a < b)
print (resultado)
```

```
True
```

En Python, una expresión que es cierta tiene el valor 1, y una expresión que es falsa tiene el valor 0.

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

```
True
```

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

```
True
```

1.3.2 Operadores lógicos

Además de los operadores relacionales, tenemos los operadores lógicos. Existen 3 tipos de operadores lógicos: and (y), or (ó), y not (no). Por ejemplo:

x > 0 and x < 10

es verdadero sólo si *x* es mayor que 0 **Y también** es menor que 10.

n%2 == 0 or n %3 == 0

es verdadero si cualquiera de las condiciones es verdadera, o sea, si el número es divisible por 2 **o** por 3. O sea, podemos leer la línea anterior como **n** divido 2 es igual a 0 **ó** n dividido 3 es igual a 0.

Finalmente, el operador **not** niega una expresión booleana, 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
and	¿se cumple a y b?	True **and** False	False
or	¿se cumple a o b?	True **or** False	True
not	No a	**not** True	False

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 (verdadero), 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 complicado

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

```
True
```

Desglocemos un poco este 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 booleana 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, siempre es recomendable utilizar paréntesis para demarcar la expresión sobre la que deseamos operar.