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 2015-10-25 12:00

LICENCIA CC BY-SA 4.0

1 Unidad 2: Tipos básicos

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

Anteriormente vimos 2 de los 3 tipos básicos que utiliza python:

- Numéricos
- Cadenas de caracteres
- Lógicos

1.1 Numéricos

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

1.1.1 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` (de integer, entero). Por ejemplo:

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

```
int
```

1.1.2 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 x 10(-3) = 0.6 x 0.001 = 0.0006

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

```
float
```

1.1.3 Complejos

Los números complejos son aquellos que tienen parte 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
```

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") o triples ("'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
```

1.3 Lógicos

Por último, nos queda el tipo básico *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.

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
```

```
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, que resulta verdadera (`True`), dado que 10 es mayor que 2. Al tratarse de una operación lógica, el resultado siempre será de tipo `bool`.

```
type(c)
```

```
bool
```

1.4 Operadores

1.4.1 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
==	¿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

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

<=	¿es a menor o igual que b?	5 <= 5	True
>=	¿es a mayor o igual que b?	2 >= 3	False

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 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
```

El tipo como hemos visto, es `bool`:

```
type(d)
```

```
bool
```

Además de cadenas también podemos comparar números y valores lógicos:

Números

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

```
True
```

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
```

1.4.2 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)
```

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. El resto de la división es cero cuando el dividendo y el divisor son múltiplos.

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`, 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
```

Desglosemos 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 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, siempre es recomendable utilizar paréntesis para demarcar la expresión sobre la que deseamos operar.