GOBIERNO DE ESPAÑA
MINISTERIO DE HACIENDA Y FUNCIÓN PÚBLICA
INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Introducción a la programación en Python

Tema 2. Tipos de variables y operador de base

Autor: Jesús Moreno Jabato

Contenido

1. Tipos de datos

1.1. Números y operaciones matemáticas

1.1.1. Enteros y decimales

Python distingue entre números enteros y decimales. Al escribir un número decimal, el separador entre la parte entera y la parte decimal es un punto.

```
print(2)
2
print(6.5)
6.5
```

Si se escribe una coma como separador entre la parte entera y la decimal, Python no lo entiende como separador, sino como una pareja de números (concretamente, lo entiende como una tupla de dos elementos, un tipo de datos que se comenta en la lección sobre tuplas).

```
print(3,8)
3 8
```

Si se escribe un número con parte decimal 0, Python considera el número como número decimal.

```
print(39.0)
39.0
```

Se puede escribir un número decimal sin parte entera, pero lo habitual es escribir siempre la parte entera:

```
print(.35)
0.35
```

En algunos casos será necesario especificar qué tipo de número necesitamos, por lo que debemos conocer los tipos que hay en Python:

- Enteros (int).
- Enteros largos (long).
- Reales (float).
- Complejos (complex).

1.1.2. Números complejos

Python puede hacer cálculos con número complejos. La parte imaginaria se acompaña de la letra "j".

```
print(1 + 1j + 2 + 3j)
(3+4j)
print((1+1j) * 1j)
(-1+1j)
```

La letra "j" debe ir acompañada siempre de un número y unida a él.

1.1.3. Las cuatro operaciones básicas

Las cuatro operaciones aritméticas básicas son la suma (+), la resta (-), la multiplicación (*) y la división (/).

Al hacer operaciones en las que intervienen números enteros y decimales, el resultado es siempre decimal. En el caso de que el resultado no tenga parte decimal, Python escribe 0 como parte decimal para indicar que el resultado es un número decimal:

```
print(4.5 * 3)
13.5
print(4.5 * 2)
9.0
```

Al sumar, restar o multiplicar números enteros, el resultado es entero.

```
print(1 + 2)
3
print(3 - 4)
-1
print(5 * 6)
30
```

Al dividir números enteros, el resultado es siempre decimal, aunque sea un número entero. Cuando Python escribe un número decimal, lo escribe siempre con parte decimal, aunque sea nula.

```
print(9 / 2)
4.5
print(9 / 3)
3.0
```

Dividir por cero genera un error:

```
print(5 / 0)
Traceback (most recent call last):
ZeroDivisionError: division by zero
Process finished with exit code 1
```

Al realizar operaciones con decimales, los resultados pueden presentar errores de redondeo:

```
print(100 / 3)
33.333333333333336
```

Cuando en una fórmula aparecen varias operaciones, Python las efectúa aplicando las reglas usuales de prioridad de las operaciones (primero multiplicaciones y divisiones, después sumas y restas).

```
print(1 + 2 * 3)
7
print(10 - 4 * 2)
2
```

En caso de querer que las operaciones se realicen en otro orden, se deben utilizar paréntesis.

```
print((5+8) / (7-2))
2.6
```

Debido a los errores de redondeo, dos operaciones que debieran dar el mismo resultado pueden dar resultados diferentes:

```
print((4 * 3 / 5))
2.4
print((4 / 5 * 3))
2.4000000000000004
```

Se pueden escribir sumas y restas seguidas, pero no se recomienda hacerlo porque no es una notación habitual:

```
print(3 + - + 4)
-1
print(3 + - + - 4)
7
```

Lo que no se puede hacer es escribir multiplicaciones y divisiones seguidas:

```
print(3 * / 4)
SyntaxError: invalid syntax
```

1.1.4. Cociente de una división

El cociente de una división se calcula en Python con el operador //.

El resultado es siempre un número entero, pero será de tipo entero o decimal dependiendo del tipo de los números empleados (en caso de ser decimal, la parte decimal es siempre cero). Por ejemplo:

```
print(10 // 3)
3
print(10 // 4)
2
print(20.0 // 7)
2.0
print(20 // 6.0)
3.0
```

El operador cociente // tiene la misma prioridad que la división:

```
print(26 // 5 / 2)
2.5
print( (26//5) / 2)
2.5
print( 26 // (5/2))
10.0
print(26 / 5 // 2)
2.0
print( (26/5) // 2)
2.0
print(26 / (5//2))
13.0
```

1.1.5. Cadenas de texto

Una cadena es una secuencia inmutable de caracteres Unicode, delimitada por comillas.

Comillas simples y dobles

Las cadenas de texto se pueden delimitar con comillas simples (') o con comillas dobles ("):

```
print('Esto es una cadena')
Esto es una cadena
print("Esto es una cadena")
Esto es una cadena
```

En Python las comillas dobles y las comillas simples son completamente equivalentes, pero en otros lenguajes de programación no lo son.

Las cadenas se deben cerrar con las mismas comillas con las que se abrieron, de lo contrario estaremos cometiendo un error de sintaxis:

```
print("Esto es una cadena')
SyntaxError: EOL while scanning string literal
```

Comillas triples

Las comillas triples permiten que las cadenas ocupen más de una línea:

```
print("""Esto es una cadena
que ocupa
  varias líneas""")

Esto es una cadena
que ocupa
  varias líneas
```

Comillas dentro de comillas

Se pueden escribir comillas simples en cadenas delimitadas con comillas dobles y viceversa:

```
print("Las comillas simples ' delimitan cadenas.")
Las comillas simples ' delimitan cadenas.
print('Las comillas dobles " delimitan cadenas.')
Las comillas dobles " delimitan cadenas.
```

Caracteres especiales

Los caracteres especiales empiezan por una barra (\):

- Comilla doble: \"

```
print("Las comillas dobles \" delimitan cadenas.")
Las comillas dobles " delimitan cadenas.
```

- Comilla simple: \'

```
print('Las comillas simples \' delimitan cadenas.')
Las comillas simples ' delimitan cadenas.
```

- Salto de línea: \n

```
print("Una línea\nOtra línea")
Una línea
Otra línea
```

- Tabulador: \t

```
print("1\t2\t3")
1       2       3
```

2. Variables

2.1. Variables en Programación

En Programación las variables están asociadas a variables concretos. Cada lenguaje de programación tiene su forma de implementar el concepto de variable, por lo que lo que se explica a continuación es válido para muchos lenguajes de programación, aunque otros lenguajes de programación permiten otras posibilidades.

Una variable se puede entender como una especie de caja en la que se puede guardar un valor (por ejemplo, un valor numérico). Esa caja suele corresponder a una posición de memoria en la memoria del ordenador.

Las variables se representan también mediante letras o palabras completas: x, y, a, b, nombre, apellidos, edad, etc.

Cuando en esos lenguajes de programación escribimos la instrucción siguiente, lo que estamos pidiendo al programa es que guarde el valor 2 en una "caja" y que a la caja le llame a.

```
a = 2
```

En esos lenguajes, el símbolo igualdad (=) hay que entenderlo como una asignación, no como una igualdad matemática.

En esos lenguajes, no estaría permitido escribir lo siguiente:

```
2 = a
x + 3 = 5
```

La información que se guarda en una variable puede ser de muchos tipos:

- Números (enteros, decimales, imaginarios…).
- Cadenas de texto (una sola letra o más letras, del juego de caracteres ASCII occidental o del juego de caracteres Unicode…).
- Conjuntos de números o texto (arrays, matrices, listas, tuplas...).
- Estructuras más complicadas (vectores, diccionarios...).

Cada tipo de información se almacena de forma distinta, por lo que existen diferentes tipos de variables para cada tipo de información.

Algunos lenguajes de programación (C, C++, Java) exigen que antes de utilizar una variable se defina el tipo de información que se va a guardar en esa variable.

Otros lenguajes de programación (Python, PHP) no lo exigen y es el intérprete del lenguaje el que decide el tipo de variable a utilizar en el momento que se guarda la información.

Los lenguajes que requieren definir los tipos de las variables se denominan lenguajes tipificados y los que no se denominan lenguajes no tipificados. Python es un lenguaje no tipificado, aunque en Python 3 se ha introducido la posibilidad de indicar los tipos de datos de los argumentos de las funciones y de los valores devueltos.

2.2. Contantes

Las constantes, a diferencia de las variables, conservan su valor a lo largo del programa. La única diferencia en su definición es que en Python, las constantes, deben ser definidas en mayúsculas.

```
# Definimos la variable ld_radio
ld_radio = 5
# Definimos la constante PI
LD_PI = 3.1415
# Longitud de la circunferencia
print (2 * LD_PI * ld_radio)
# Asignamos un nuevo radio
ld_radio = 2.45
# Longitud de la circunferencia
print (2 * LD_PI * ld_radio)
31.415000000000003
15.393350000000002
```

2.3. Variables en Python

En algunos lenguajes de programación, las variables se pueden entender como "cajas" en las que se guardan los datos, pero en Python las variables son "etiquetas" que permiten hacer referencia a los datos (que se guardan en unas "cajas" llamadas objetos).

Python es un lenguaje de programación orientado a objetos y su modelo de datos también está basado en objetos.

Para cada dato que aparece en un programa, Python crea un objeto que lo contiene. Cada objeto tiene:

- un identificador único (un número entero, distinto para cada objeto). El identificador permite a Python referirse al objeto sin ambigüedades.
- un tipo de datos (entero, decimal, cadena de caracteres, etc.). El tipo de datos permite saber a Python qué operaciones pueden hacerse con el dato.
- un valor (el propio dato).

Las variables en Python no guardan los datos, sino que son simples nombres para poder hacer referencia a esos objetos. Si escribimos la instrucción siguiente:

```
a = 2
```

- crea el objeto "2". Ese objeto tendrá un identificador único que se asigna en el momento de la creación y se conserva a lo largo del programa. En este caso, el objeto creado será de tipo número entero y guardará el valor 2.
- asocia el nombre a al objeto número entero 2 creado.

De esta forma, al describir la instrucción anterior no habría que decir 'la variable a almacena el número entero 2', sino que habría que decir 'podemos llamar a al objeto número entero 2'. La variable a es como una etiqueta que nos permite hacer referencia al objeto "2", más cómoda de recordar y utilizar que el identificador del objeto.

Por otro lado, en Python se distingue entre objetos mutables y objetos inmutables:

- Los objetos inmutables son objetos que no se pueden modificar. Por ejemplo, los números, las cadenas y las tuplas son objetos inmutables.
- Los objetos mutables son objetos que se pueden modificar. Por ejemplo, las listas y los diccionarios son objetos mutables.

En el caso de los objetos inmutables (como los números) no hay mucha diferencia entre considerar la variable como una caja o como una etiqueta, pero en el caso de los objetos mutables (como las listas) pensar en las variables como cajas puede llevar a error.

Aunque no es obligatorio, se recomienda que el nombre de la variable esté relacionado con la información que se almacena en ella, para que sea más fácil entender el programa.

El nombre de una variable debe empezar por una letra o por un guion bajo (_) y puede seguir con más letras, números o guiones bajos.

Los nombres de variables no pueden incluir espacios en blanco.

Los nombres de variables pueden contener cualquier carácter alfabético (los del alfabeto inglés, pero también ñ, ç o vocales acentuadas), aunque se recomienda utilizar únicamente los caracteres del alfabeto inglés.

Los nombres de las variables pueden contener mayúsculas, pero tenga en cuenta que Python distingue entre mayúsculas y minúsculas (en inglés se dice que Python es "case-sensitive").

Cuando el nombre de una variable contiene varias palabras, se aconseja separarlas con guiones bajos para facilitar la legibilidad, aunque también se utiliza la notación camelCase, en las que las palabras no se separan pero empiezan con mayúsculas (salvo la primera palabra).

Para visualizar una variable se usará el comando print

```
Print (a)
```

Para más información para comprender qué es un Objeto y el concepto de clase:

http://courseware.url.edu.gt/Facultades/Facultad%20de%20Ingenier%C3%ADa/Ingenier%C3%ADa%20en%20Inform%C3%A1tica%20y%20Sistemas/Segundo%20Ciclo%202011/Introducci%C3%B3n%20a%20la%20Programaci%C3%B3n/Objetos%20de%20aprendizaje/Unidad%202B/Unidad%202B/cul_es_la_diferencia_entre_una_clase_y_un_objeto.html

2.4. Definir una variable

Las variables en Python se crean cuando se definen por primera vez, es decir, cuando se les asigna un valor por primera vez.

Para asignar un valor a una variable se utiliza el operador de igualdad (=). A la izquierda de la igualdad se escribe el nombre de la variable y a la derecha el valor que se quiere dar a la variable.

En el ejemplo siguiente se almacena el número decimal 2,5 en una variable de nombre x. Fíjate en que los números decimales se escriben con punto (.) y no con coma (,).

```
x = 2.5
```

La variable se escribe siempre a la izquierda de la igualdad. Si se escribe al revés, Python genera un mensaje de error:

```
2.5 = x
SyntaxError: can't assign to literal
```

Para que Python muestre el valor de una variable, basta con escribir su nombre:

```
x = 2.5
print(x)
2.5
```

Si una variable no se ha definido previamente, escribir su nombre genera un mensaje de error:

```
x = -10
print(y)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    y
NameError: name 'y' is not defined
```

Una variable puede almacenar números, texto o estructuras más complicadas. Si se va a almacenar texto, el texto debe escribirse entre comillas simples (') o dobles ("), que son equivalentes. A las variables que almacenan texto se les suele llamar cadenas (de texto).

```
nombre = "Pepe Palotes"
print (nombre)
'Pepe Palotes'
```

Si no se escriben comillas, Python supone que estamos haciendo referencia a otra variable (que, si no está definida, genera un mensaje de error):

```
nombre = Pepe
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    nombre = Pepe
NameError: name 'Pepe' is not defined
```

```
nombre = Pepe Palotes
SyntaxError: invalid syntax
```

Para visualizar una variable dentro de un texto se puede hacer de varias maneras, una es haciendo una cadena de strings.

```
nombre = "Pepe Palotes"
print ("Tu nombre es",nombre,", muy bonito por cierto")
Tu nombre es Pepe Palotes , muy bonito por cierto
```

Y otra es usando el comando formated string literal, f-string.

```
nombre = "Pepe Palotes"
print (f"Tu nombre es {nombre}, muy bonito por cierto")
Tu nombre es Pepe Palotes, muy bonito por cierto
```

2.5. Borrar una variable

La instrucción “del” borra completamente una variable.

```
nombre = "Pepito Conejo"
del nombre
print(nombre)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    nombre
NameError: name 'nombre' is not defined
```

2.6. Tipos de variables

En el apartado anterior hay definiciones de algunos de los tipos de variables que hay en Python:

Aunque se definan de forma similar los tipos de variables de números decimales, números enteros y cadenas (una o más letras). Para Python no es lo mismo ya que por ejemplo, dos números se pueden multiplicar pero dos cadenas no (curiosamente, una cadena sí que se puede multiplicar por un número).

Por tanto, estas tres definiciones de variables no son equivalentes:

```
fecha = 1997
fecha = 1997.0
fecha = "1997"
fecha = [27, "octubre", 1997]
```

- En el primer caso la variable fecha está almacenando un número entero
- En el segundo fecha está almacenando un número decimal
- En el tercero fecha está almacenando una cadena de cuatro letras.
- En el cuarto fecha está almacenando una lista (un tipo de variable que puede contener varios elementos ordenados).

Este ejemplo demuestra también que se puede volver a definir una variable. Python modifica el tipo de la variable automáticamente.

3. Booleanos

3.1. Tipos booleanos: True y False

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

En Python cualquier variable 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", 2020])
True
bool({27, "octubre", 2020})
True
```

3.2. 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. 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
```

3.3. 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
```

3.4. 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.

Pyhon 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
```

3.5. Funciones type

En Python podemos usar la función type para saber el tipo de un valor:

```
type(2)
class 'int'
type(2+3j)
class 'complex'
```