Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

3

Variables. Quiero decir... nombres

Índice de contenidos

Introducción

En este tema ya me pongo serio y voy a entrar en materia de la buena.

Vamos a hablar de las variables, esos elementos que forman parte de un programa y que se utilizan para guardar y acceder a los datos.

No obstante, el tema guarda una sorpresa. Más bien será una de las primeras lecciones que aprendas como Pythonista: En Python, las variables no existen.

Esto es algo que muchos programadores en Python no saben. ¿Quieres descubrir por qué?

Doy paso al tema.

Una primera aproximación a las variables en Python

Las variables son uno de los dos componentes básicos de cualquier programa.

En su esencia, un programa está compuesto por datos e instrucciones que manipulan dichos datos. Normalmente, los datos se almacenan en memoria (memoria RAM) para que podamos acceder a ellos.

Entonces, ¿qué es una variable? Una variable es una forma de identificar, de forma sencilla, un dato que se encuentra almacenado en la memoria del ordenador.

Imagina que una variable es un contenedor en el que se almacena o se guarda un dato, el cuál, puede cambiar durante el flujo del programa. Estos contenedores (las variables) permiten acceder fácilmente y en cualquier momento a los datos para ser manipulados y transformados.

Imagina un programa que va mostrando, paso a paso, el resultado de sumar los números del 1 al 3:

```
>>> suma = 1
>>> print(suma)
1
>>> suma = suma + 2
>>> print(suma)
3
>>> suma = suma + 3
>>> print(suma)
6
```

El ejemplo es un poco tonto, pero quiero que veas que el programa está formado por datos (guardados en la variable `suma`) e instrucciones que manipulan dichos datos. En este caso, sumar a la variable `suma` 2 y 3, respectivamente, y mostrar por pantalla el valor de dicha variable.

Como has comprobado, si a la función `print()` se le pasa como argumento una variable, se muestra su valor por pantalla.

Asignar un valor a una variable en Python

Tal y como hemos visto en el ejemplo anterior, para guardar un valor (un dato) en una variable se utiliza el operador de asignación `=`.

En la instrucción de asignación se ven involucradas tres partes:

- El operador de asignación `=`.
- Un identificador o nombre de variable, a la izquierda del operador.
- Un literal, una expresión, una llamada a una función o una combinación de todos ellos a la derecha del operador de asignación.

Ejemplos:

```
# Asigna a la variable <a> el valor 1
a = 1

# Asigna a la variable <a> el resultado de
# la expresión 3 * 4
a = 3 * 4

# Asigna a la variable <a> la cadena de
# caracteres 'Pythonista'
a = 'Pythonista'
```

Cuando asignamos un valor a una variable por primera vez, se dice que en ese lugar se define e inicializa la variable. En un script o programa escrito en Python, podemos definir variables en cualquier lugar de este. Sin embargo, es una buena práctica definir las variables que vayamos a utilizar al principio.

Si intentas usar una variable que no ha sido definida/inicializada previamente, el intérprete te mostrará un error:

```
>>> print(a)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'a' is not defined
```

Como te comenté en el tema de introducción, Python es un lenguaje tipado dinámicamente. Por esta razón, a diferencia de otros lenguajes como *C* o *Java*, no se especifica el tipo de dato que guarda una variable cuando se declara. Como sabrás, los tipos de datos definen qué conjunto de valores puede tomar una variable y qué operaciones se pueden realizar sobre los mismos.

El siguiente tema está dedicado, precisamente, a los tipos de datos. No obstante, te hago aquí un adelanto de cuáles son los tipos de datos básicos de Python: los numéricos (*enteros*, *reales* y *complejos*), los booleanos (*True*, *False*) y las cadenas de caracteres.

Veamos un ejemplo:

```
a = 1       # a es de tipo int
b = 'Hola'  # b es de tipo string
```

El hecho de que Python sea tipado dinámicamente permite que, en cualquier momento, una variable que es de un tipo pueda convertirse en una variable de otro tipo diferente (al final del tema verás que esto no es así realmente):

```
a = 1       # a es de tipo int
a = 'Hola'  # Ahora a es de tipo string
```

Se dice que Python (el intérprete) infiere el tipo de dato al que pertenece una variable en tiempo de ejecución, es decir, es cuando se ejecuta el programa, cuando

conoce su tipo de dato y qué operaciones pueden llevarse a cabo con él.

Literales

Tal y como te he mencionado en un apartado anterior, a una variable se le puede asignar un literal, una expresión, una llamada a una función o una combinación de todos ellos.

Ahora bien, ¿qué es un literal? Ya hemos visto ejemplos de literales en los apartados anteriores. Un literal no es más que un valor en crudo que se puede asignar directamente a una variable o formar parte de una expresión.

En el siguiente ejemplo se asigna a la variable `a` el literal `1`:

```
a = 1
```

El literal `1` representa el valor numérico del número entero 1.

Existen varios tipos de literales:

- Numéricos (enteros, reales y complejos): `1`, `3`, `3.14`, `-1`, `3.14j`, ...
- Cadenas de caracteres: `'Hola'`, `'Me gusta Python'`, ...
- Booleanos: `True` y `False`.
- El literal especial `None`. `None` significa ausencia de valor y es muy utilizado en programación.

Asignar un valor a múltiples variables

Ahora vas a descubrir cómo asignar un mismo valor a múltiples variables a la vez. Si quieres asignar a varias variables un mismo dato, puedes usar la siguiente estructura:

```
>>> a = b = c = 1
>>> print(a)
1
>>> print(b)
1
>>> print(c)
1
```

Asignar múltiples valores a múltiples variables

También es posible inicializar varias variables con un valor diferente cada una del siguiente modo:

```
>>> a, b, c = 1, 2, 3
>>> print(a)
1
>>> print(b)
2
>>> print(c)
3
```

No obstante, te desaconsejo que lo hagas así porque el código es menos legible.

Variables en Python: La verdad

Para terminar este tema, quiero que te olvides de todo lo que te he explicado en las secciones anteriores, jaja.

En Python las variables no existen.

Sí, en Python no existen las variables como tal y ahora entenderás por qué.

Para empezar, te diré que **en Python todo es un objeto**. Sí, todavía no hemos visto el apartado de programación orientada a objetos con Python, pero ten presente que en Python todo es un objeto.

¿Significa esto que, al igual que en *Java*, todo el código debe ser encapsulado en una clase? No. Significa que, internamente, en la implementación de Python, todo está representado mediante clases y objetos. Para que lo entiendas: un literal es un objeto; una función es un objeto; un módulo es un objeto; una clase es un objeto; por supuesto, un objeto es un objeto, ...

Todo ello implica que cada uno de los elementos mencionados arriba contenga las propiedades que suelen tener los objetos, es decir, atributos y métodos.

Juanjo, ¿en serio esto es así? Como te lo digo. Créeme que esto es así y ahora lo verás.

¿Por qué te he contado todo esto? Para que entiendas realmente qué es una variable en Python. Mejor dicho, para que entiendas por qué en Python las variables no existen.

Observa el siguiente ejemplo en el que se asigna el valor 1 a una “variable”.

```
>>> contador = 1
```

Entonces, en el ejemplo anterior, ¿`contador` no es una variable? La mayoría de lenguajes lo llamarían variable. De hecho, muchos artículos de Python se referirán a `contador` como una variable, pero en realidad **es un nombre o identificador**.

En lenguajes como *C*, `contador` representa un lugar para almacenar valores de tipo entero. Sería correcto decir que la variable entera `contador` contiene el valor `1`. Es decir, las variables son una especie de contenedor de valores.

En Python no es así. Es un error decir que `contador` "contiene" el valor `1`. Por el contrario, `contador` **es un nombre enlazado o asociado al objeto que representa al número 1**. Asignar 1 a `contador` simplemente significa "Quiero poder referirme a este objeto como `contador` ". Por tanto, **en lugar de variables** (en el sentido clásico), **Python tiene nombres y enlaces**.

Entonces, ¿qué ocurre cuando a una “variable” `a` le asigno el valor 1?

```
a = 1
```

Básicamente, que la variable `a` hace referencia, está relacionada o está asociada al objeto que representa al número entero con valor 1. `a` es un nombre con el que identificar a dicho valor. Si ahora creas una nueva variable `b` y le asignas `a`, `b` está haciendo referencia al mismo objeto que la variable `a`. En definitiva, `a` y `b` hacen referencia al mismo objeto y, por tanto, están "enlazadas" a la misma dirección de memoria. Esto no es así en otros lenguajes, pero esto es Python, jaja. En otros lenguajes `a` y `b` harían referencia a direcciones de memoria diferentes.

Para que lo entiendas mejor te voy a presentar la función `id()` que viene con Python. La función `id()` devuelve un identificador único del objeto que se le pasa como parámetro. Este identificador es un entero el cuál se garantiza que es único e inmutable durante toda la vida del objeto en memoria.

! En la implementación de Python *CPython*, este identificador es realmente la dirección en memoria del objeto.

Veamos todo lo anterior con un ejemplo:

```
# a y b hacen referencia al objeto que
# representa al entero 1.
# Referencian a la misma dirección de
# memoria.
>>> a = 1
>>> b = a
>>> id(1)
4299329328
>>> id(a)
4299329328
>>> id(b)
4299329328

# b es enlazado con el objeto que
# representa el entero 2.
# a y b referencian a diferentes
# direcciones de memoria.
# a mantiene la referencia al entero 1.
>>> b = 2
>>> id(a)
4299329328
>>> id(b)
4299329360

# Se asocia a con el valor de b.
# a y b referencian al mismo objeto.
>>> a = b
>>> id(a)
4299329360
>>> a
2
```

En definitiva, todo en Python es un objeto. Todo lo que se pueda (legalmente) colocar en el lado derecho del signo igual es (o crea) un objeto en Python.

El número 1 del ejemplo anterior es un objeto y no simplemente un valor como en otros lenguajes. ¿Sigues sin creértelo?

Comprueba lo siguiente por tu cuenta:

```
>>> a = 1
>>> print(a.__add__)
<method-wrapper '__add__' of int object
at 0x1093cbf30>
```

Si 1 no fuera un objeto, no tendría el método `__add__`.

En Python, puedes ver todos los elementos que componen una clase con la función `dir()`.

Bueno, pues con esto terminamos este tema dedicado a las vari..., a los nombres. Seguramente, a lo largo del curso utilizaré la palabra variable. Tú ya sabes que,

realmente, me estoy refiriendo al nombre con el que se identifica a un objeto concreto.