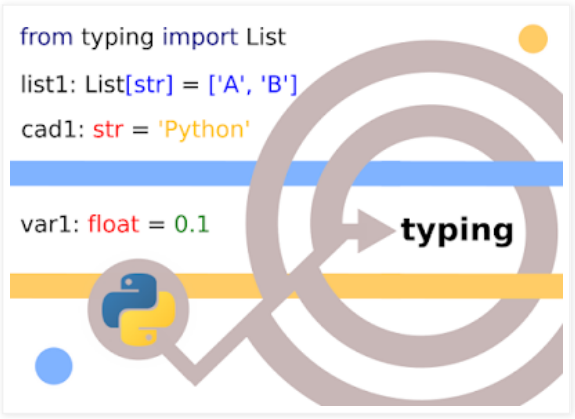
# Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)



**miércoles, 15 de abril de 2020**

## Anotaciones de tipos: typing




Una de las características del lenguaje Python es que el tipado que usa es dinámico, es decir, permite que una variable pueda cambiar su tipo durante la ejecución de un programa. La razón es que los tipos dependen del valor que tenga asignado dicha variable en un momento dado, no de una propiedad de la variable en sí.

Este rasgo de flexibilidad en el uso de las variables que conlleva no estar obligados a anotar los tipos son ventajas que a veces inquietan a los más ortodoxos, en especial, a los que desarrollan también con lenguajes de tipado estático como Java, C y C++. Los argumentos que aducen son la falta de claridad en el código y la perdida de tiempo intentando identificar el tipo de los valores de las variables y de otros objetos, en particular, en proyectos de cierta envergadura.

Con el paso del tiempo, después de debates no libres de controversias, estas razones han sido tomadas en consideración. Y aunque existían ya módulos externos con el mismo propósito, es a partir de Python 3.5 cuando se añade el módulo **typing** a la librería estándar, para permitir anotar los tipos de forma nativa (PEP 484).

De momento, anotar los tipos es opcional y la descripción que se haga no se aplica en tiempo de ejecución. Esto significa que la anotación de una variable puede indicar que es de tipo **str** pero terminar como **float**, en definitiva, algo que siempre ha podido suceder en Python.

Y entonces ¿Qué finalidad tiene anotar los tipos? Por un lado informativo, actuando como parte de la documentación que se añade al código y, por otro, como la sintaxis es estricta puede ser utilizada por herramientas de terceros con la capacidad de comprobar el uso adecuado de las variables de acuerdo a sus anotaciones de tipos, en el desarrollo de un proyecto. Entre los verificadores de tipo disponibles se encuentran Mypy, Pyre y Pytype. Mypy es la herramienta de referencia desarrollada por Dropbox.

El siguiente código Python no solo es posible, además, es aceptable. Implementarlo con lenguajes de tipado estático conlleva tener que declarar más de una variable:

```
repetir = 2
print(repetir)  # 2
print(type(repetir))  # class 'int'

repetir = repetir * 'Typing'
print(repetir)  # TypingTyping
print(type(repetir))  # class 'str'
```

A continuación, mostramos cómo se anotan los tipos más comunes en Python.

### Variables y constantes

**Buscar**

Buscar

**Python para impacientes**

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

**Anexos**

Guía urgente de MySQL
Guía rápida de SQLite3

**Entradas + populares**

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario ( GUI ) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

Para describir o anotar el tipo de las **variables** y **constantes** se agrega "**: tipo**" a continuación del nombre (el tipo puede ser **str**, **int**, **float**, **bool**, etc.) y, después, si es necesario se asigna un valor tras el signo igual '**=**'. En el siguiente ejemplo se describen dos variables: **repetir** de tipo **int** e inicializada con el valor **3** y **cadena** de tipo **str** y no inicializada. Lo recomendado es declarar todas las variables con sus tipos al principio del código para aumentar su legibilidad:

```
repetir: int = 3
cadena: str

cadena = repetir * 'Hola'
print(repetir)   # 3
print(cadena)   # HolaHolaHola
```

El intérprete de Python almacena las anotaciones de tipo en el atributo especial **__annotations__**:

```
print(__annotations__)  # {'repetir': class int, 'cadena': class str}
```

## Funciones

Para anotar el tipo de los argumentos de una **función** se agrega ": **tipo**" a continuación del nombre de cada argumento y, después, si es necesario se asigna el valor por defecto tras el signo '**=**'. Para describir el tipo del valor de retorno se añade " **-> tipo**" después de los argumentos y antes de los dos puntos "**:**" del final:

```
def precio(entradas: int, festivo: bool=False) ->int:
    if festivo:
       return entradas * 10
    else:
       return entradas * 8

print(precio(3, True))  # 30
print(precio(4))  # 32
```

## Clases

Para anotar los tipos en una **clase** se aplica a sus **atributos** la misma sintaxis que a las variables y a las constantes; y a sus **métodos** la misma que a las funciones:

```
class Juego:
    tiempo: int = 40

    def __init__(self, nom: str, nivel: int, jug: int = 1) -> None:
        self.nom= nom
        self.nivel = nivel
        self.jug = jug

    def duracion(self) -> int:
        return self.jug * Juego.tiempo

partida = Juego('Ajedrez', 1, 2)
print(partida.duracion())  # 80
```

## Anotaciones para tipos complejos

Para anotar tipos más complejos como **listas**, **tuplas**, **diccionarios**, **conjuntos** y otros es necesario utilizar el módulo **typing** e importar los tipos que correspondan, como en el ejemplo que sigue. Para describir el tipo de una lista se importa **List** y entre corchetes se anota el tipo "**[tipo]**" de los elementos a contener, en este caso **str**. Para el tipo del diccionario se importa **Dict** y entre corchetes "**[tipo_clave, tipo_valor]**" se indica los tipos de las **claves** y de los **valores** separados por una coma ",". Para describir un conjunto se importa **Set** y entre corchetes se indica el tipo "**[tipo]**" de los valores a contener. También, se pueden utilizar en otro ámbito, como en la función del ejemplo **imprime_rios()**, para indicar que el argumento **rios** recibe una lista de cadenas **str**:

```
from typing import List, Dict, Set

apellidos: List[str] = ['Alcantara', 'Alonso', 'Blanco']
referencias: Dict[str, int] = {'Mesa': 121, 'Silla': 485}
```

que permiten obtener de distintos modos números a...

```
serie: Set[int] = {1, 1, 1, 2, 2, 3, 3, 5, 5, 5, 5, 5, 6}


def imprime_rios(rios: List[str]) -> None:
    for rio in rios:
        print(rio)

imprime_rios(['Guadalquivir', 'Tinto', 'Odiel', 'Segura'])
```

### Alias

Los **alias** permiten la creación de anotaciones de tipo con denominaciones propias para mejorar la comprensión del código. En el ejemplo siguiente se define el tipo **Color** como una tupla de tres valores de tipo **int** para expresar la cantidad de **rojo**, **verde** y **azul** asociado a un determinado color. Después, en la función **pintar()** se anota junto al argumento **color** el tipo **Color** que se corresponde con dicha tupla:

```
from typing import Tuple

Color = Tuple[int, int, int]

def pintar(color: Color) -> None:
    r: int
    v: int
    a: int
    r, v, a = color
    print('Color Rojo:', r, 'Verde:', v, 'Azul:', a)

pintar((100, 200, 120))
```

### Funciones con múltiples valores de retorno

Cuando una función devuelve múltiples valores el tipo del retorno se anota como una tupla con los tipos de sus valores separados por comas, o por un tipo personalizado basado en la misma construcción: **Tuple[tipo, tipo, ...]**

```
from typing import Tuple

Coordenada = Tuple[int, int]

def coordenada_inicial() -> Coordenada:
    return 0, 0

print(coordenada_inicial())
```

### Anotaciones para variables con valores de distinto tipo

Para variables que pueden tener valores de distinto tipo existen las anotaciones predefinidas **Optional** y **Union**. **Optional** se utiliza con variables que pueden ser de un tipo concreto o de ninguno (**None**). Y **Union** es apropiado para variables cuyos valores pueden ser de tipos diferentes, excepto **None**.

En la función **representantes()** tanto el argumento **votos** como el valor de retorno son de tipo **Optional[int]**. Por ello, tanto el valor del argumento **votos** como el que devuelve la función pueden ser de tipo **int** o **None**.

```
from typing import Optional

Votos = Optional[int]
Representantes = Optional[int]

def representantes(votos: Votos) -> Representantes:
    if votos:
        return votos // 5000
    else:
        return None

print(representantes(None))  # None
print(representantes(3409))  # 0
print(representantes(11231))  # 2
```

En la función **recuento()** el valor de retorno es de tipo **Optional[str, float]**. Esto significa que el valor que retorna la función puede ser de tipo **str** o **float**: en el ejemplo si el valor del argumento **inicio** es **False** se considera que el escrutinio no ha comenzado y devuelve la cadena '**Escrutinio no iniciado**'. En cambio, si el valor de **inicio** es **True** devuelve el porcentaje escrutado como un valor de tipo **float**.

```
from typing import Union

Recuento = Union[str, float]

def recuento(inicio: bool, actual: int, final: int) -> Recuento:
    if not inicio:
        return 'Escrutinio no iniciado'
    else:
        return round((actual * 100 / final), 2)

print(recuento(False, 0, 0))  # Escrutinio no iniciado
print(recuento(True, 4560, 9800))  # 46,53
```

## Mypy

Entre los verificadores de anotaciones de tipo destaca **Mypy**, un proyecto iniciado por Jukka Lehtosalo en el que ha participado Guido Van Rossum. Esta herramienta comprueba que no existan incoherencias de tipo en uno o más archivos de código fuente. Sin más preámbulos, mostramos su uso:

Para instalar Mypy:

```
$ pip install mypy
```

Para ver cómo funciona Mypy utilizaremos el siguiente ejemplo que incluye un error en la anotación de tipo del valor de retorno:

esfera.py:

```
from math import pi

def vol_esfera(radio: float) -> int:
    return round(4/3 * pi * radio ** 3, 2)

print(vol_esfera(2))  # 33,51 metros cúbicos
```

Para comprobar el código, ejecutar:

```
$ mypy esfera.py
```

Salida:

```
typando1.py:4: error: Incompatible return value type (got "float",
expected "int")
Found 1 error in 1 file (checked 1 source file).
```

La salida sugiere que el tipo del valor de retorno debe ser **float**. A continuación, realizamos este cambio en esfera.py...:

```
from math import pi

def vol_esfera(radio: float) -> float:
    return round(4/3 * pi * radio ** 3, 2)

print(vol_esfera(2))  # 33,51 metros cúbicos
```

Y volvemos a ejecutar Mypy para confirmar la solución aplicada:

```
$ mypy esfera.py
```

Salida:

```
Success: no issues found in 1 source file
```

Ya no hay errores de anotaciones de tipos en el código. Para verificar el código y ampliar la información que se ofrece utilizar el argumento **-v** y para conocer el resto de opciones disponibles, el argumento **-h**.

Para finalizar, recomendamos ver el código fuente de los módulos typing y mypy dado la cantidad y variedad de objetos que pueden describirse y consultar la documentación oficial.

**Relacionado**:

- Docstrings
- Programas con estilo en Python
- Probando el código fuente

Publicado por Pherkad en 9:39

Etiquetas: Python

Entrada más reciente Inicio Entrada antigua