Artículo

Funciones como objetos

David Aroesti 11 de Diciembre de 2019

Funciones como objetos

Una de las características más poderosas de Python es que todo es un objeto, incluyendo las funciones. Las funciones en Python son "ciudadanos de primera clase".

Esto, en sentido amplio, significa que en Python las funciones:

- Tienen un tipo
- Se pueden pasar como argumentos de otras funciones
- Se pueden utilizar en expresiones
- Se pueden incluir en varias estructuras de datos (como listas, tuplas, diccionarios, etc.)

Argumentos de otras funciones

Hasta ahora hemos visto que las funciones pueden recibir parámetros para realizar los cómputos que definen. Algunos de los tipos que hemos pasado son tipos simples como cadenas, números, listas, etc. Sin embargo, también pueden recibir funciones para crear abstracciones más poderosas. Veamos un ejemplo:

```
def multiplicar_por_dos(n):
    return n * 2

def sumar_dos(n):
    return n + 2

def aplicar_operacion(f, numeros):
    resultados = []
    for numero in numeros:
        resultado = f(numero)
        resultados.append(resultado)

>>> nums = [1, 2, 3]
>>> aplicar_operacion(multiplicar_por_dos, nums)
[2, 4, 6]

>>> aplicar_operacion(sumar_dos, nums)
[3, 4, 5]
```

Funciones en expresiones

Una forma de definir una función en una expresión es utilizando el keyword `lambda`. `lambda` tiene la siguiente sintaxis: `lambda <vars>: <expresion>`.

Otro ejemplo interesante es que las funciones se pueden utilizar en una expresión directamente. Esto es posible ya que como lo hemos platicado con anterioridad, en Python las variables son simplemente nombres que apuntan a un objeto (en este caso a una función). Por ejemplo:

```
sumar = lambda x, y: x + y

>>> sumar(2, 3)
5
```

Funciones en estructuras de datos

Las funciones también se pueden incluir en diversas estructuras que las permiten almacenar. Por ejemplo, una lista puede guardar diversas funciones a aplicar o un diccionario las puede almacenar como valores.

```
def aplicar_operaciones(num):
    operaciones = [abs, float]

    resultado = []
    for operacion in operaciones:
        resultado.append(operacion(num))

    return resultado

>>> aplicar_operaciones(-2)
[2, -2.0]
```

Como pudimos ver, las funciones son objetos muy versátiles que nos permiten tratarlas de diversas maneras y que nos permiten añadir capas adicionales de abstracción a nuestro programa.

Compártenos cómo te imaginas que estas capacidades de Python te pueden ayudar a escribir mejores programas.