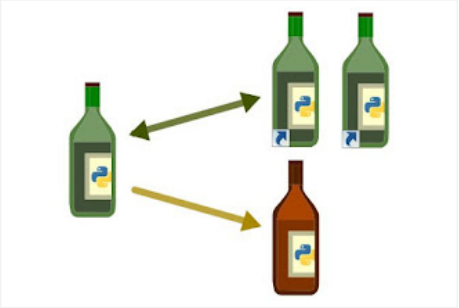
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 22 de septiembre de 2018

Copia superficial y profunda de variables



Identificadores de objetos

En Python cuando se declara una variable el intérprete le asigna un identificador único interno que se corresponde con un número entero que el sistema utiliza para diferenciarla de otras variables (y de otros objetos) en la memoria. Cuando dicha variable se asigna a otra el intérprete no crea una copia del dato en otro área diferente de la memoria sino que declara un enlace que conecta la nueva variable con el valor de la primera, compartiendo ambas un mismo identificador (sucede también cuando se declara una nueva variable asignando un valor que ya exista en memoria). Dicho identificador se corresponde con un número entero que se genera en cada ejecución y se puede obtener con la función **id()**.

En el ejemplo siguiente se declaran cinco variables y al consultar sus identificadores se obtienen solo dos diferentes: uno se utiliza con las variables (*var1*, *var2* y *var3*) y el otro con las restantes (*var4* y *var5*) porque en ambos casos cada grupo de variables comparte un mismo valor.

```
var1 = 1
var2 = var1
var3 = var1
var4 = [[1, 2], [var1, var2]]
var5 = var4

id(var1)  # 8805376
id(var2)  # 8805376
id(var3)  # 8805376
id(var4)  # 139944016265480
id(var5)  # 139944016265480
```

En adelante, si ninguna variable sufriera cambios seguirían con los mismos identificadores o, si alguna cambiara, puede suceder que se mantengan los mismos o que se originen otros diferentes asociados a nuevos espacios de memoria. Depende del tipo de estructura de datos y del tipo de operación. Por ejemplo, cuando se realiza una asignación no se gestionan del mismo modo las variables que contienen datos únicos que aquellas que contienen listas, diccionarios, conjuntos o instancias de clases; y tampoco se maneja igual una simple lista de datos que otra en la que se incluyen referencias a otros objetos.

A continuación se asigna el valor *0* a la variable *var1*. Esta operación supone que *var1* ya no comparta su valor con *var2* y *var3* y que el sistema genere un nuevo identificador para *var1*:

```
var1 = 0

var1  # 0
var2  # 1
var3  # 1
```

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

```
id(var1)  # 8805344 (nuevo identificador)
id(var2)  # 8805376
id(var3)  # 8805376
```

Sin embargo, en la siguiente asignación no sucede igual. Los identificadores de *var4* y *var5* siguen siendo los mismos y modificar *var4* es equivalente a modificar *var5*:

```
var4[0][1] = 7

var4  # [[1, 7], [1, 1]]
var5  # [[1, 7], [1, 1]]

id(var4)  # 139944016265480
id(var5)  # 139944016265480
```

Estos mecanismos del intérprete para referenciar variables y duplicar la información se utilizan junto a otros para optimizar el uso de los recursos. Y suceden sin que un programador tenga que ser, en principio, muy consciente de ello. Puede ser normal con lenguajes de alto nivel pero hay que tener cuidado porque pueden producir efectos inesperados. Además, están los casos en los que es necesario tratar las variables por separado, es decir, en los que no se quiere que al modificar el valor de una variable cambie el valor de otras.

Las funciones copy() y deepcopy() del módulo copy

Para evitar errores se recomienda el uso de las funciones que proporciona el módulo **copy** de la librería estándar de Python. Este módulo consta de funciones para duplicar variables y otros objetos con distinto nivel del profundidad: **copy()** para **copias superficiales** y **deepcopy()** para **copias profundas**.

La diferencia entre una copia superficial y una profunda es trascendental para objetos que se componen de otros objetos como listas, diccionarios, conjuntos o instancias de clase. En una copia superficial de este tipo de objetos se crea un nuevo objeto y después, en lo posible, se insertan las referencias de los objetos contenidos en el original. Sin embargo, en una copia profunda después de la creación del nuevo objeto se incorporan copias de los objetos contenidos en el original de forma recursiva. Por ello, un objeto copiado de forma superficial no es completamente independiente del original como sí ocurre con uno obtenido de una copia profunda, implicando esta última un proceso de creación más lento.

Para copias superficiales además de la función **copy()** se pueden utilizar la funciones **list()**, **dict()** y **set()** empleadas para declarar listas, diccionarios y conjuntos, respectivamente.

En el ejemplo siguiente se utilizan las funciones **copy()** y **deepcopy()**. En él se declaran una lista simple (*lista1*) de dos elementos, la lista *lista2* que anida dos listas (la primera es una referencia a *lista1* y la segunda contiene los elementos *[3, 4]*). Finalmente, a partir de *lista2* se crean las listas *lista3* y *lista4* mediante una copia superficial y otra profunda, respectivamente.

Más adelante se muestra el contenido de las cuatro listas y el valor de sus identificadores, incluyéndose los de las referencias a *lista1* en *lista2*, *lista3* y *lista4* que coincide con el primer elemento (posición 0) de cada una de las listas.

Podemos observar que ninguna de las listas comparte un mismo identificador pero que en la referencias a *lista1* que se hace en el resto de listas los identificadores coinciden en *lista2* y *lista3* pero que difiere con el de *lista4* que se obtuvo mediante una copia profunda.

```
import copy
lista1 = [1, 2]
lista2 = [lista1, [3, 4]]
lista3 = copy.copy(lista2)
lista4 = copy.deepcopy(lista2)

lista1  # [1, 2]
lista2  # [[1, 2], [3, 4]]
lista3  # [[1, 2], [3, 4]]
lista4  # [[1, 2], [3, 4]]

id(lista1)  # 139895660929992
id(lista2)  # 139895624466376
id(lista3)  # 139895641593544
id(lista4)  # 139895624474760
id(lista2[0])  # 139895660929992
id(lista3[0])  # 139895660929992
id(lista4[0])  # 139895624466120
```

Para comprobar el funcionamiento de los dos tipos de copias se asigna *9* al segundo elemento de *lista1* y dicho cambio queda reflejado en *lista2* y *lista3* pero no en *lista4* que se obtuvo mediante una copia profunda.

que permiten obtener de distintos modos números a...

Archivo

septiembre 2018 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
lista1[1] = 9

lista1  # [1, 9]
lista2  # [[1, 9], [3, 4]]
lista3  # [[1, 9], [3, 4]]
lista4  # [[1, 2], [3, 4]]
```

Para finalizar cambiamos los valores de los segundos elementos de cada lista contenida en *lista2*. En el primer caso el valor *5* modifica *lista1*, *lista2* y *lista3* por compartir el mismo identificador y en el segundo el valor *-5* modifica la segunda lista de *lista2* y *lista3*, podemos intuir, por compartir también un mismo identificador; y quedando otra vez excluida la *lista4* por haberse obtenido su copia con **deepcopy()**.

```
lista2[0][1] = 5
lista2[1][1] = -5

lista1  # [1, 5]
lista2  # [[1, 5], [3, -5]]
lista3  # [[1, 5], [3, -5]]
lista4  # [[1, 2], [3, 4]]
```

Relacionado:

- Palabras reservadas. Variables. Cadenas
- Cadenas, listas, tuplas, diccionarios y conjuntos (set)
- Programación orientada a objetos

Ir al índice del tutorial de Python

Publicado por Pherkad en 3:19

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua