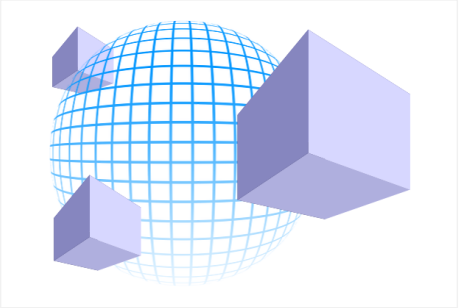
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

domingo, 15 de marzo de 2015

Diccionarios de variables locales y globales



Funciones locals() y globals()

Las funciones **locals()** y **globals()** devuelven *diccionarios* con las variables locales y globales que pueden utilizarse en un programa.

Cada diccionario equivale a un *espacio de nombre* (***namespace***) donde Python gestiona las variables propias de un ámbito determinado. Si el diccionario es local el ámbito se refiere al espacio de nombre de una función o una *clase*; y si el diccionario es global el ámbito se corresponde con el módulo.

En el siguiente ejemplo se muestran los dos diccionarios en diferentes ámbitos para poder analizar después su contenido:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

from math import pi
global1 = 1

def funcion1(x, y):
    global global1
    total = x + y + global1
    global1 = 2
    print("funcion1. Dicc. locales.:", locals())
    print("funcion1. Dicc. globales:", globals())
    return total

class clase1:
    z = 3
    print("clase1. Dicc. locales.:", locals())
    print("clase1. Dicc. Globales:", globals())
    def __init__(self):
        print("clase1: método __init__")

    def __call__(self):
        print("clase1: método __call__")

def main():
    a = 5
    b = 10

    if "funcion1" in globals():
        if callable(globals()["funcion1"]):
            print("Rtdo funcion1: ", globals()["funcion1"](a, b))

    objeto = clase1()
```

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

```
    if "objeto" in locals():
        if callable(locals()["objeto"]):
            locals()["objeto"]

    locals()["a"] = 20
    globals()["b"] = 20
    print("main(). Dicc. locales.:", locals())
    print("main(). Dicc. globales:", globals())
    print("a: ", a, "b:", b)
    return 0

if __name__ == '__main__':
    main()
```

En los diccionarios se mantienen las variables definidas por el usuario, las variables que tienen referencias a funciones y módulos, las variables propias de Python como **__doc__**, **__main__**, **__file__**, **__module__**, etc. Entre las variables del ejemplo se incluye una de tipo **global** llamada ***global1*** para hacer un seguimiento de su valor en distintas partes del programa.

Contenido de los diccionarios en funcion1:

- Función:

```
def funcion1(x, y):
    global global1
    total = x + y + global1
    global1 = 2
    print("funcion1. Dicc. locales.:", locals())
    print("funcion1. Dicc. globales:", globals())
    return total
```

- Diccionarios:

funcion1. Dicc. locales.: {'y': 10, 'total': 16, 'x': 5}

funcion1. Dicc. globales: {'__name__': '__main__', '__file__': 'localsglobals.py', 'global1': 2, 'pi': 3.141592653589793, '__builtins__': <module 'builtins' (built-in)>, '__package__': None, 'funcion1': <function funcion1 at 0xb70578e4>, '__cached__': None, '__doc__': None, '__loader__': <_frozen_importlib.SourceFileLoader object at 0xb70b160c>, 'main': <function main at 0xb7057854>, 'clase1': <class '__main__.clase1'>, '__spec__': None}

En el diccionario **locals()** están todas las variables que se utilizan dentro de la propia función, con sus valores correspondientes.

En el diccionario **globals()** está la variable global ***global1*** con el valor **2**, la constante ***pi*** con su valor, que es importada al comienzo desde el módulo ***math***, la variable ***__file__*** que contiene el nombre del archivo Python (***localsglobals.py***), las variables que contienen la referencia a la propia función y a la clase ***clase1***, la variable que contiene las *cadenas de documentación* o *docstrings*, ***__doc__***, que en este caso está vacía y otras variables globales.

Contenido de los diccionarios en clase1:

- Clase:

```
class clase1:
    z = 3
    print("clase1. Dicc. locales.:", locals())
    print("clase1. Dicc. Globales:", globals())
    def __init__(self):
        print("clase1: método __init__")

    def __call__(self):
        print("clase1: método __call__")
```

- Diccionarios:

clase1. Dicc. locales.: {'__qualname__': 'clase1', '__module__': '__main__', 'z': 3}

clase1. Dicc. Globales: {'__name__': '__main__', '__loader__': <_frozen_importlib.SourceFileLoader object at 0xb70b160c>, 'global1': 1, '__cached__': None,

que permiten obtener de distintos modos números a...

Archivo

marzo 2015 (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

'pi': 3.141592653589793, '__builtins__': <module 'builtins' (built-in)>, '__package__': None, '__file__': 'localsglobals.py', 'funcion1': <function funcion1 at 0xb70578e4>, '__spec__': None, '__doc__': None}

En el diccionario **locals()** está la variable ***z*** que se declara en la clase, el nombre de la variable que contiene el nombre cualificado o ***__qualname__*** de la clase y la referencia del módulo en la variable ***__module__***.

En el diccionario **globals()** llama la atención la variable ***global1*** que en este caso mantiene el valor inicial, **1**. También, está la constante ***pi*** con su valor y no aparecen las referencias a la función main(), ni a la propia clase.

Contenido de los diccionarios en main():

- Función main():

```
def main():
    a = 5
    b = 10
    if "funcion1" in globals():
        if callable(globals()["funcion1"]):
            print("Rtdo funcion1: ", globals()["funcion1"](a, b))

    objeto = clase1()
    if "objeto" in locals():
        if callable(locals()["objeto"]):
            locals()["objeto"]

    locals()["a"] = 20
    globals()["b"] = 20
    print("main(). Dicc. locales.:", locals())
    print("main(). Dicc. globales:", globals())
    print("a: ", a, "b:", b)
    return 0
```

- Diccionarios:

main(). Dicc. locales.: {'objeto': <__main__.clase1 object at 0xb7113f4c>, 'a': 5, 'b': 10}

main(). Dicc. globales: {'__name__': '__main__', 'clase1': <class '__main__.clase1'>, 'funcion1': <function funcion1 at 0xb70fe974>, '__spec__': None, '__file__': 'localsglobals.py', '__package__': None, '__builtins__': <module 'builtins' (built-in)>, '__loader__': <_frozen_importlib.SourceFileLoader object at 0xb715860c>, '__cached__': None, 'pi': 3.141592653589793, 'global1': 2, '__doc__': None, 'b': 20, 'main': <function main at 0xb70fe89c>}

En el diccionario **locals()** están las variables ***a*** y ***b*** declaradas en main() con sus respectivos valores y, también, una variable llamada ***objeto*** con una referencia a la clase ***clase1*** que es la utilizada para su creación: ***objeto = clase1()***.

En el diccionario **globals()** se encuentra la variable global ***global1*** con el valor **2**. Con respecto al resto de variables se repiten los mismos nombres de variables que en el diccionario de la función ***funcion1***, excepto la variable ***b*** que tiene el valor ***20*** y que se comenta más adelante.

A cualquier diccionario se puede acceder para consultar un valor utilizando el nombre de la variable correspondiente; pero sólo en el caso del diccionario global se podrá cambiar el valor de una variable. Debemos tener en cuenta que si accedemos con posterioridad al valor de una variable siempre tendrá prioridad el valor que tenga la variable en el diccionario local.

En la función main() se "intentan" modificar los valores de ***a*** y ***b*** con:

locals()["a"] = 20
globals()["b"] = 20

La modificación en el diccionario local no es posible. En cambio, en el diccionario global se ha podido cambiar el valor de la variable ***b***. Sin embargo, si imprimimos esta variable obtendremos *el valor* ***5;*** que es el que tiene en el diccionario local. Esto es así porque los valores de las variables del diccionario local tienen prioridad sobre los existentes en el diccionario global.

Hay un diccionario o *espacio incorporado* que es global a todos los módulos. Si Python no encuentra un nombre en el diccionario local ni en el global accederá al incorporado. Si tampoco hay éxito se producirá una excepción del tipo ***NameError***.

Llamadas de retorno. La función callable()

En el diccionario global del ejemplo se comprueba si existe la función ***función1*** y (antes de llamarla), también, se verifica con la función **callable()** si dicho objeto puede ser llamado:

```
if "funcion1" in globals():
    if callable(globals()["funcion1"]):
        print("Rtdo funcion1: ", globals()["funcion1"](a, b))
```

Esta característica que permite ejecutar funciones que se crean dinámicamente, sin conocerse a priori sus nombres, se denomina *llamadas de retorno*..

A continuación, se muestra otro caso similar en el que se crea un objeto. A continuación, se comprueba si existe dicho objeto en el diccionario local y después de verificar con la función **callable()** que el objeto se puede invocar, se le llama:

```
objeto = clase1()
if "objeto" in locals():
    if callable(locals()["objeto"]):
        locals()["objeto"]
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 16:54

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua