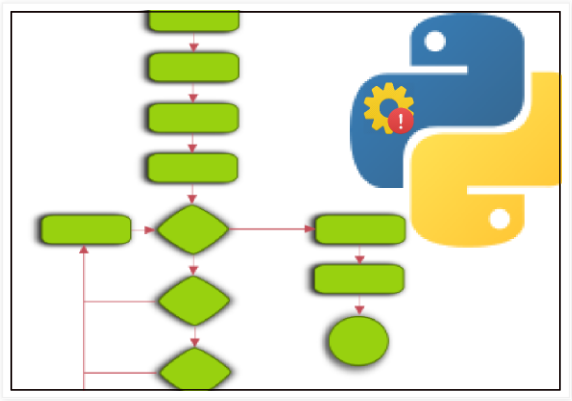
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

viernes, 6 de febrero de 2015

Solucionando errores con el depurador



El módulo de depuración pdb

El módulo **pdb** de la librería estándar es una ayuda importante para los programadores Python. Se utiliza para analizar si un programa hace lo que se espera de él. En esencia facilita la detección de errores de codificación y agiliza su corrección.

Antes de continuar, es necesario entender que un depurador no se ocupa de los errores de sintaxis pues esta tarea la tiene encomendada el propio intérprete; y a dicha función se entrega incondicionalmente mientras se ejecuta un programa (***.py***), o bien, durante el proceso de compilación cuando se desea obtener el correspondiente archivo compilado o *bytecode* (***.pyc***).

Fundamentalmente, el módulo **pdb** permite ejecutar un programa, instrucción a instrucción; detenerlo en un punto específico (***breakpoint***) para poder examinar el valor de alguna variable o cambiar su valor antes de continuar; y/o ejecutar alguna instrucción para verificar algún resultado. Todo ello lo hace teniendo el control del programa en todo momento, hasta que termine su ejecución de la forma prevista o hasta que su finalización sea forzada.

Tipos de errores

Durante el desarrollo de un programa podemos encontrarnos, básicamente, con tres tipos de errores:

- Los ***errores de interpretación o compilación*** que son producidos por instrucciones mal escritas o por un uso incorrecto de la sintaxis del lenguaje. En Python, estos errores tienen que ser solucionados para que un programa pueda ser ejecutado.
- Los ***errores de ejecución*** que se producen mientras se ejecuta un programa, a pesar de estar todo el código escrito conforme a las reglas del lenguaje. Pueden suceder por inconsistencia de los datos, al intentar dividir por 0, por no poder acceder a un dispositivo, por otros errores externos al programa, etc.
- Y los ***errores de lógica*** que son aquellos derivados de una errónea codificación y que implican que el programa no alcance el propósito para el que fue concebido.

Uso del depurador

Para diferenciar algunos errores de los comentados y ver cómo funciona el depurador lo mejor será basarnos en un caso práctico.

A continuación, vamos a escribir un programa muy simple llamado (***cambia.py***) que cuando se ejecute pedirá que se introduzca una cadena de caracteres. La introducción terminará al presionar la tecla [*return*]. Después, el programa buscará en la cadena introducida todos los caracteres que sean asteriscos "*" y los sustituirá por comas ",". Este programa incorpora dos errores diferentes que vamos a estudiar:

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...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

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

entrada = input("Escribe algo: ")
entrada.replace("*", ",")
print("Salida: ", entrada))
```

Probemos a ejecutar el código desde la línea de comandos:

$ python3 cambia.py

El avezado intérprete nos avisa que tenemos un error en la línea del ***print()***, en la número 6, donde aparece un paréntesis de cierre de más. Y por este motivo el programa no se puede ejecutar:

```
  File "cambia.py", line 6
    print("Salida: ", entrada))
                              ^
SyntaxError: invalid syntax
```

Este error de sintaxis podemos reproducirlo si intentamos compilar el código con:

$ python3 -m py_compile cambia.py

Desde la versión 3.2 de Python cuando se compila un programa, si no hay errores, se crea al mismo nivel del archivo del programa una carpeta llamada ***__pycache__*** y, en ella, se genera su versión compilada o ***bytecode***.

En el nombre del archivo compilado aparece la versión del intérprete con que se compiló y su extensión, que como ya hemos comentado, será .pyc. En este caso el nombre será "*cambia.cpython-34.pyc*" y contendrá el literal "cpython".

CPython es el nombre que se da a la implementación oficial y original de Python, (la que instalamos desde Python.org), escrita en lenguaje C, que es también el propio intérprete del ***bytecode***. Como seguramente conoce el lector hay otras implementaciones de Python como: *Jython, PyPy, IronPython*.

La razón de que aparezca la versión del intérprete en el nombre del archivo compilado está en la posibilidad de tener instaladas varias versiones diferentes de Python en un mismo sistema y en que cada ***bytecode*** está ligado a la versión del intérprete que lo generó.

El archivo compilado o ***bytecode*** no podemos editarlo pero sí ejecutarlo desde la línea de comandos:

$ python3 cambia.cpython-34.pyc

Después de comentar algo sobre la compilación, lo siguiente será editar el programa fuente y arreglar el error quitando el paréntesis de más. Después, volveremos a ejecutar con:

$ python3 cambia.py

Y como ya no hay ningún error de sintaxis el programa comenzará su ejecución y pedirá que escribamos algún texto. A continuación, escribiremos: "hola* buenos días" (sin las comillas) y terminaremos con la tecla [*return*]. Finalmente, el programa mostrará una salida incorrecta (el asterisco se tenía que haber sustituido por una coma y eso no ha ocurrido) y terminará su ejecución. Así que el programa no hace lo que debe (error de lógica) y, sin embargo, en este caso "nadie" nos avisa del fallo:

Escribe algo: hola* buenos días
Salida: hola* buenos días

En este tipo de situaciones es donde conviene utilizar un **depurador** de código. Tengamos en cuenta que este programa es muy corto pero en uno con varios cientos de líneas podemos encontrarnos con dificultades.

Buscando el error

A continuación, volveremos a editar el programa y añadiremos al principio la siguiente línea que importa el módulo de depuración:

import pdb

Y después del ***input()*** insertamos otra línea que servirá para que el programa se detenga en ese lugar cuando sea ejecutado:

pdb.Pdb().set_trace()

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

febrero 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

Después de los últimos cambios el código quedará así:

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

import pdb

entrada = input("Escribe algo: ")
pdb.Pdb().set_trace()
entrada.replace("*", ",")
print("Salida: ", entrada)
```

Si ejecutamos e introducimos "Hola* buenos días" y [return] el depurador detendrá el programa y mostrará la línea siguiente que se va a ejecutar:

Escribe algo: Hola* buenos días
> /home/antonio/Local/Blog/blog-python3/Nuevo/depurar.py(8)<module>()
-> entrada.replace("*", ",")
(Pdb)

Ahora, si escribimos a continuación de **(Pdb)** el nombre de la variable ***entrada*** el depurador retornará su valor actual: "Hola* buenos días":

(Pdb) entrada
'Hola* buenos días'
(Pdb)

En este punto si presionamos la tecla "***n***" (*next*) y [*return*] ejecutaremos la instrucción señalada con la flecha y el depurador se detendrá de nuevo en la línea siguiente:

(Pdb) n
> /home/antonio/Local/Blog/blog-python3/Nuevo/depurar.py(9)<module>()
-> print("Salida: ", entrada)
(Pdb)

Si volvemos a preguntar por el valor de la variable ***entrada*** comprobaremos que justo antes de la línea del ***print()*** no ha cambiado ¿Por qué?:

(Pdb) entrada
'Hola* buenos días'

En este punto podemos deducir que el error está en la línea donde se utiliza el método ***replace(),*** que aunque realiza los cambios, éstos no son reasignados a la variable ***entrada*** y se pierden. Lo correcto hubiera sido codificarlo así:

entrada = entrada.replace("*", ",")

Si arreglamos el error y ejecutamos de nuevo comprobaremos que ya funciona correctamente.

Escribe algo: Hola* buenos días
Salida: Hola, buenos días

Otra prueba de depuración que podemos realizar consiste en modificar el valor de la variable ***entrada*** cuando el programa esté detenido y, después, con el comando "***c***" (*continue*) seguiremos su ejecución para ver qué sucede:

(Pbd) entrada = "Buenas tardes* América"

Otros comandos que se pueden utilizar con el depurador:

quit (q)
Finaliza la depuración.

list (l)
Lista líneas de código indicando el punto de parada.

step (s)
Ejecuta el programa paso a paso.

continue (c)
Continua ejecución hasta el final o hasta el siguiente punto de interrupción.

return (r)
Cuando se ejecuta una función continua la ejecución hasta ***return***.

nombrevariable o display var1, var2
Muestra el valor de las variables indicadas.

var2 = var1 + int(valor)
Declarar variables auxiliares a partir de otras existentes.

help
Lista todos los comandos disponibles del depurador.

Para agregar depuración desde la línea de comandos sin necesidad de importar el módulo pdb:

$ python3 -m pdb cambia.py

Para concluir, comentar que el entorno de desarrollo ***IDLE*** cuenta con un depurador integrado con posibilidad de ejecutar instrucción a instrucción, insertar puntos de interrupción y permite ver la pila de llamadas.

Relacionado:

- Facilitando la depuración de programas con breakpoint()
- El depurador PuDB
- Editar y depurar scripts (IPython)

Ir al índice del tutorial de Python

Publicado por Pherkad en 11:52

Etiquetas: depurador, Python3

Entrada más reciente Inicio Entrada antigua