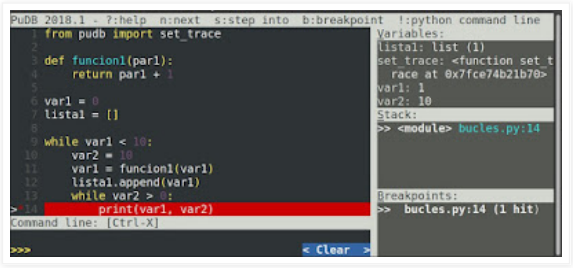
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 6 de octubre de 2018

El depurador PuDB



PuDB es un depurador ligero de código para Python con una interfaz de texto que proporciona todo lo necesario para probar programas y encontrar los errores de lógica que puedan producirse mientras se ejecutan.

Durante la ejecución del código se aplica color a su sintaxis para facilitar su lectura y se muestra información en tiempo real de las variables (y de otros objetos), de la pila de ejecución y de los puntos de parada que se establezcan para el análisis. Todo ello, distribuido en áreas separadas que favorecen su seguimiento. También, existe un área a la que es posible acceder en cualquier momento para ejecutar comandos Python o acceder a uno de los Shell disponibles (ipython, ptpython, bpython, etc).

La navegación entre las distintas áreas y el modo de interactuar con el depurador se hace mediante la pulsación de teclas lográndose que la tarea de depuración sea fácil y rápida.

Instalación de PuDB

Instalación con pip:

```
$ pip3 install pudb
```

Uso básico de PuDB

Para explicar el funcionamiento del depurador utilizaremos el código del siguiente ejemplo que incluye una función **funcion1**, a todas luces innecesaria, y varias variables que modifican su valor a medida que se ejecutan dos bucles **while**. Se trata de ejecutar el programa y hacer un seguimiento de sus distintos objetos para localizar un error y subsanarlo. El programa que cuenta con dos contadores (**var1** y **var2**) debe presentar en pantalla sus valores mostrando como **var1** se va incrementando de 1 en 1 (desde 1 a 10) cada vez que **var2** completa un ciclo que va desde 10 a 1 (decreciendo de 1 en 1). También hay una variable de tipo lista **lista1** a la que se va añadiendo el valor que va tomando **var1** en cada ciclo, que se muestra al finalizar.

bucles.py:

```
def funcion1(par1):
    return par1 + 1

var1 = 0
lista1 = []

while var1 < 10:
    var2 = 10
    var1 = funcion1(var1)
    lista1.append(var1)
    while var2 > 0:
        print(var1, var2)
        var2=-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...

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

```
print(lista1)
```

Si ejecutamos el código con:

```
$ python3 bucles.py
```

Comprobaremos que la salida que se obtiene es errónea, que no se corresponde con lo esperado. En ella **var1** se incrementa de 1 en 1 pero **var2** se mantiene siempre con el valor 10. Según lo comentado con anterioridad antes de que **var1** cambie al siguiente valor, **var2** tiene que ir desde 10 a 1, decreciendo de 1 en 1.

Salida errónea:

```
1 10
2 10
3 10
4 10
5 10
6 10
7 10
8 10
9 10
10 10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

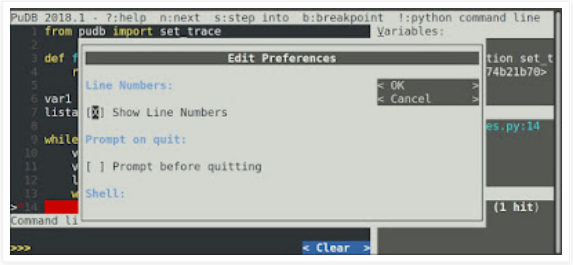
Para comenzar la depuración del programa insertar al principio del código la siguiente línea:

```
from pudb import set_trace
```

Después, iniciar el depurador con:

```
$ pudb3 prueba.py
```

La primera vez que se inicia el depurador se muestra la ventana de bienvenida con información general de la versión. Para avanzar presionar **[return]**.



Después hay que configurar las preferencias del depurador en la ventana **Edit Preferences**: básicamente, entre sus opciones hay que seleccionar si se van a mostrar o no los números de líneas junto al código, si habrá que confirmar o no la salida del depurador, el Shell Python que se utilizará de forma predeterminada, el tema o combinación de colores de la interfaz, el orden de la pila de ejecución y el modo de mostrar las variables.

Para desplazarse entre las opciones utilizar las fechas de desplazamiento del cursor **[arriba]** y **[abajo]** y para seleccionar o deseleccionar una opción la **[barra espaciadora]**. Cuando se hayan establecido todas las opciones, para aceptarlas, presionar la flecha **[derecha]** para situar el cursor en la ventana de las opciones **OK** y **Cancel**. Seleccionar **OK** y confirmar con **[return]**.

Más adelante, cuando se desee acceder a la ventana **Edit Preferences** para cambiar alguna opción, presionar **[Control+p]**. Toda la información de la configuración se guarda en el archivo **~/.config/pudb/pudb.cfg**.

A continuación, se mostrará en pantalla la interfaz de PuDB con el código del programa a depurar en el área mayor; a su derecha se muestran las áreas para seguimiento de **Variables**, **Stack** y **Breakpoints**; y en la parte inferior el área **Command line** para ejecutar comandos Python.

Antes de iniciar la depuración podemos insertar uno o más puntos de parada en las líneas de código que consideremos más adecuadas para la exploración. Para ello, utilizar las teclas de flecha **[arriba]** y **[abajo]** para desplazar el selector de línea y presionar **[b]** para insertar o eliminar un punto de parada.

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

Archivo

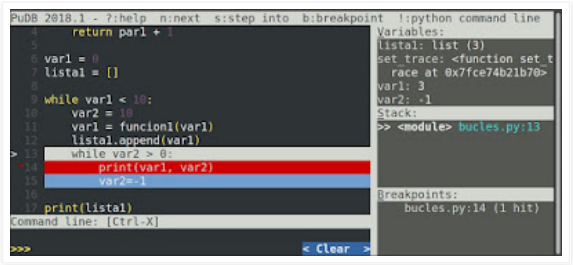
octubre 2018 (2)

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



En este caso desplazar el selector de línea a la **línea 14** ("**print(var1, var2)**") utilizando la tecla de flecha **[abajo]** y presionar la tecla **[b]** para insertar un punto de parada (la línea aparecerá con el color rojo de fondo). Después, para iniciar la ejecución desde la primera línea hasta alcanzar dicho punto presionar **[c]**.

En este momento el programa se encuentra detenido en la **línea 14**. En la ventana **Variables** aparecen ya los valores actuales de las variables e información de otros objetos: **var1 = 1**, **var2 = 10**, etc. A partir de ahora podemos avanzar línea a línea presionando la tecla **[n]**. A medida que pulsamos y completamos ciclos observamos en el área **Variables** que el valor de **var2** cambia su valor de **10** a **-1**, alternando estos valores en vez de ir decreciendo de 1 en 1 como estaba previsto. Concretamente, al pasar por la **línea 15** se asigna erróneamente el valor -1, cuando en realidad habría que restar 1 a **var2**. Para subsanarlo salir del depurador presionando la tecla **[q]** y cambiar el código de la **línea 15** que es '**var2=-1**' por '**var2-=1**'. Después, probar la ejecución y el resultado debe ser correcto. Si suprimimos el punto de parada actual presionando la tecla **[b]** en la **línea 14** y después presionamos la tecla **[c]** el programa se ejecutará hasta el final.

Una ventaja de PuDB es que "recuerda" los puntos de parada para futuras sesiones porque se guardan en el archivo **~/.config/pudb/saved-breakpoints**. Estos puntos de parada también pueden establecerse insertando en el código la función **set_trace()**.

Durante la depuración para cambiar al área **Variables** presionar **[V]**. En este área se puede modificar el modo de visualizar un objeto. Por ejemplo, en vez de visualizar el número de elementos que contiene una lista se pueden ver los propios elementos.

Para cambiar entre las áreas se pueden usar las teclas de desplazamiento del cursor en cualquier dirección. Por ejemplo, al presionar la tecla de flecha **[abajo]** cada vez que se alcance el límite inferior de un área se pasará al área inmediata de abajo. Esto funciona en cualquier sentido según se presione flecha **[arriba]**, **[abajo]**, **[izquierda]** o **[derecha]**. Otra alternativa para cambiar de área directamente es mediante las teclas siguientes:

- [C] - Code
- [V] - Variables
- [S] - Stack
- [B] - Breakpoints

También, desde cualquier área se puede pasar al área **Command Line** para ejecutar comandos Python presionando **[Control-x]** o pasar de ésta al área del código con la misma combinación de teclas.

Por último, si se desea iniciar desde PuDB el Shell configurado en las opciones de preferencia presionar **[!]**. Y para ver información de ayuda de otros usos del teclado pulsar **[?]**.

Hay distintas alternativas para depurar programas Python pero para los amantes del Terminal y, en particular, de las interfaces de texto en PuDB encontrarán un buen aliado.

Relacionado:

- Solucionando errores con el depurador
- Facilitando la depuración de programas con breakpoint()
- Editar y depurar scripts (IPython)

Ir al índice del tutorial de Python

Publicado por Pherkad en 10:15

Etiquetas: depurador, Python3

Entrada más reciente Inicio Entrada antigua