TATA ELXSI

Python Scripting

Learning & Development Team

Monday, November 2, 2020

Module – 9 : PDB

Disclaimer

- This material is developed for in house training at **TATA ELXSI**.
- *TATA ELXSI is not responsible for any errors or omissions, or for the results obtained from the use of this information. All information in this material is provided "as is", with no guarantee of completeness, timeliness or of the results obtained from the use of this information...*
- *This is a training material and the information contained herein is not intended to be a source of advice or research outcome with respect to the material presented, and the information and/or documents/examples contained in this material do not constitute authors or TATA ELXSI's advice.*
- The training material therefore was solely developed for educational purposes. Currently, commercialization of the prototype does not exist, nor is the prototype available for general usage.
- **TATA ELXSI**, its staff, its students or any other participants can use it as a part of training. This material should not be found on any website.
- For the preparation of this material we have referred from the below mentioned links or books.
- Excerpts of these material has been taken where there is no copy right infringements.

Agenda

- How does the debugger work?
- Debugger Commands
- Setting (conditional) breakpoints
- Single stepping at the source line level,
- Inspection of stack frames,
- Source code listing

Debugger -- pdb

- The module pdb defines an interactive source code debugger for Python programs

- . Typical usage to run a program under control of the debugger is:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
```

- pdb.py can also be invoked as a script to debug other scripts. For example:

- python3   -m   pdb   myscript.py

Debugger -- pdb

- Command tp print the documentation on pdb

```
>>> import pdb
>>> pdb.help()
```

- Command to print the short documentation on pdb commands

```
(Pdb) help clear
cl(ear) filename:lineno
cl(ear) [bpnumber [bpnumber...]]
```

- Debugger commands

```
=================
 h(elp)
   Without argument, print the list of available commands.
```

Debugger -- pdb

- w(here)

  Print a stack trace, with the most recent frame at the bottom. An arrow indicates the "current frame", which determines the context of most commands. 'bt' is an alias for this command.

- d(own) [count]

  Move the current frame count (default one) levels down in the stack trace (to a newer frame).

- u(p) [count]

  Move the current frame count (default one) levels up in the stack trace (to an older frame).

Debugger -- pdb

- b(reak) [([filename:]lineno | function) [, condition]]

  Without argument, list all breaks.

cl(ear) filename:lineno

cl(ear) [bpnumber [bpnumber...]]

With a space separated list of breakpoint numbers, clear those breakpoints. Without argument, clear all breaks

- disable bpnumber [bpnumber ...]

  Disables the breakpoints given as a space separated list of breakpoint numbers it remains in the list of breakpoints and can be (re-)enabled.

Debugger -- pdb

- enable bpnumber [bpnumber ...]

  Enables the breakpoints given as a space separated list of breakpoint numbers

- s(tep)

  Execute the current line, stop at the first possible occasion (either in a function that is called or in the current function).

- n(ext)

  Continue execution until the next line in the current function is reached or it returns.

j(ump) lineno

Set the next line that will be executed. Only available in the bottom-most frame.

Debugger -- pdb

- l(ist) [first [,last] | .]

    List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing.

- display [expression]

    Display the value of the expression if it changed, each time execution stops in the current frame.

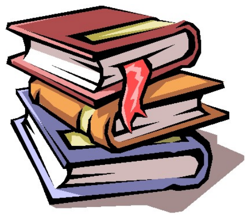
Example 1:

```
import pdb;

def add( a,b) :
    return a+b

def sub(a,b):
    return a-b

def mul(a,b):
    return a*b

def div(a,b):
    return a/b

pdb.set_trace()

val1=add(5,10)
val2=sub(5,10)
val3=div(5,10)
val4=mul(5,10)


print(val1)
print(val2)
print(val3)
print(val4)
```

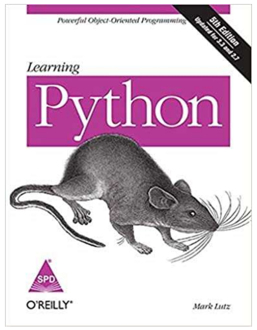
Example 2:

```
import pdb

def fun(a):
    ret1=foo(a*10)
    print("Return from foo",ret1)
    print("a in  fun is :",a)
    return ret1

def foo(a):
    ret2=bar(a*10)
    print("Return from bar",ret2)
    print("a in  bar is :",a)
    return ret2

def bar(a):
    ret3=(a*10)
    print("value in bar is ",ret3)
    print("a in  fun is :",a)
    return ret3

pdb.set_trace()
fun(10)
```

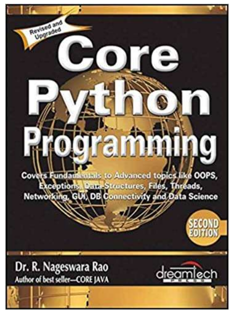
References



- **Python 3.x.x documentation**: https://docs.python.org/3/



- **Learning Python: Powerful Object-Oriented Programming: 5th Edition**


Revised and Upgraded
Core Python Programming
Covers Fundamentals to Advanced topics like OOPS, Exceptions, Data Structures, Files, Threads, Networking, GUI, DB Connectivity and Data Science
SECOND EDITION
Dr. R. Nageswara Rao
Author of best seller—CORE JAVA
dreamtech

- **Core Python Programming [**by R. Nageswara Rao (Author)]

Thank you

Learning & Development

Tata Elxsi

Bangalore

www.tataelxsi.com

TATA ELXSI