



Python Debugger – Python pdb

Difficulty Level : Medium • Last Updated : 30 Sep, 2022



Debugging in Python is facilitated by **pdb module** (python debugger) which comes built-in to the Python standard library. It is actually defined as the class Pdb which internally makes use of bdb(basic debugger functions) and cmd (support for line-oriented command interpreters) modules. The major advantage of pdb is it runs purely in the command line, thereby making it great for debugging code on remote servers when we don't have the privilege of a GUI-based debugger.

pdb supports:

- Setting breakpoints
- Stepping through code
- Source code listing
- Viewing stack traces

Starting Python Debugger

There are several ways to invoke a debugger

- To **start debugging within the program** just insert `import pdb`, `pdb.set_trace()` commands. Run your script normally, and execution will stop where we have introduced a breakpoint. So basically we are hard coding a breakpoint on a line below where we call `set_trace()`. With python 3.7 and later versions, there is a built-in function called **breakpoint()** which works in the same manner. Refer following example on how to insert `set_trace()` function.

Example1: Debugging a Simple Python program of addition of numbers using Python pdb module



Python3

```
import pdb

def addition(a, b):
    answer = a * b
    return answer

pdb.set_trace()
x = input("Enter first number : ")
y = input("Enter second number : ")
sum = addition(x, y)
print(sum)
```

Output :

```
C:\Users\admin\Desktop\CS_Stuff\pythoVS>python exppdb.py
> c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py(10)<module>()
-> x = input("Enter first number : ")
(Pdb) █
```

set_trace

In the output on the first line after the angle bracket, we have the **directory path** of our file, **line number** where our breakpoint is located, and **<module>**. It's basically saying that we have a breakpoint in exppdb.py on line number 10 at the



is stopped. That line is not executed yet. Then we have the **pdb prompt**. Now to navigate the code, we can use the following commands :

Command	Function
---------	----------

help	To display all commands
------	-------------------------

where	Display the stack trace and line number of the current line
-------	---

next	Execute the current line and move to the next line ignoring function calls
------	--

step	Step into functions called at the current line
------	--

Now, to check the type of variable, just write **whatis** and variable name. In the example given below, the output of type of x is returned as <class string>. Thus typecasting string to int in our program will resolve the error.

Example 2: Checking variable type using pdb 'whatis' command

We can use '**whatis**' keyword followed by a variable name (locally or globally defined) to find its type.

Python3

```
a = 20
b = 10

s = 0
for i in range(a):
    # this line will raise ZeroDivision error
    s += a / b
    b -= 1
```



```
(Pdb) c
Traceback (most recent call last):
  File "/usr/lib/python3.8/pdb.py", line 1705, in main
    pdb._runscript(mainpyfile)
  File "/usr/lib/python3.8/pdb.py", line 1573, in _runscript
    self.run(statement)
  File "/usr/lib/python3.8/bdb.py", line 580, in run
    exec(cmd, globals, locals)
  File "<string>", line 1, in <module>
  File "/test_2.py", line 1, in <module>
    a = 20
ZeroDivisionError: division by zero
Uncaught exception. Entering post mortem debugging
Running 'cont' or 'step' will restart the program
> test_2.py(1)<module>()
-> a = 20
(Pdb) whatis a
<class 'int'>
(Pdb) whatis b
<class 'int'>
(Pdb) whatis s
<class 'float'>
(Pdb)
```

Checking variable types using whatis command.

Finding variable type using whatis command in pdb

- **From the Command Line:** It is the easiest way of using a debugger. You just have to run the following command in terminal

```
python -m pdb exppdb.py (put your file name instead of exppdb.py)
```

This statement loads your source code and stops execution on the first line of code.

Example 3: Navigating in pdb prompt

We can navigate in pdb prompt using n (next), u (up), d (down). To debug and navigate all throughout the Python code, we can navigate using the mentioned commands.

Python3

```
a = 20
b = 10

s = 0
for i in range(a):
    s += a / b
    b -= 1
```

Output :

```
(common_env3.8) $ python -m pdb test_2.py
> test_2.py(1)<module>()
-> a = 20
(Pdb) jump 4      ← Jumped to line 4
> test_2.py(4)<module>()
-> s = 0
(Pdb) n          ← Jumped to next line (i.e. line 5)
> test_2.py(5)<module>()
-> for i in range(a): Showing contents of line 5
(Pdb) █
```

Navigate in pdb prompt using commands

Example 4: Post-mortem debugging using Python pdb module

Post-mortem debugging means entering debug mode after the program is finished with the execution process (failure has already occurred). pdb supports post-mortem debugging through the **pm()** and **post_mortem()** functions. These functions look for active trace back and start the debugger at the line in the call stack where the exception occurred. In the output of the given example, you can notice pdb appear when an exception is encountered in the program.

Python3

```
def multiply(a, b):
    answer = a * b
    return answer

x = input("Enter first number : ")
y = input("Enter second number : ")
result = multiply(x, y)
print(result)
```

Output :



```

Enter first number : 23
Enter second number : 34
Traceback (most recent call last):

  File "C:\Users\Vanshi\Desktop\gfg\untitled6.py", line 8, in <module>
    result = multiply(x, y)

  File "C:\Users\Vanshi\Desktop\gfg\untitled6.py", line 2, in multiply
    answer = a * b

TypeError: can't multiply sequence by non-int of type 'str'

In [5]: import pdb

In [6]: pdb.pm()
> c:\users\vanshi\desktop\gfg\untitled6.py(2)multiply()
1 def multiply(a, b):
----> 2     answer = a * b
      3     return answer
      4
      5

ipdb>

```

Checking variables on the Stack

All the variables including variables local to the function being executed in the program as well as global are maintained on the stack. We can use **args** (or use **a**) to print all the arguments of a function which is currently active. **p** command evaluates an expression given as an argument and prints the result.

Here, example 4 of this article is executed in debugging mode to show you how to check for variables :

```

(Pdb) next
Enter second number : 34
> c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py(8)<module>()
-> result = multiply(x, y)
(Pdb) step
--Call--
> c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py(1)multiply()
-> def multiply(a, b):
(Pdb) args
a = '23'
b = '34'
(Pdb) next
> c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py(2)multiply()
-> answer = a * b
(Pdb) p b
'34'
(Pdb)

```

checking variable values



While working with large programs, we often want to add a number of breakpoints where we know errors might occur. To do this you just have to use the **break** command. When you insert a breakpoint, the debugger assigns a number to it starting from 1. Use the **break** to display all the breakpoints in the program.

Syntax:

```
break filename: lineno, condition
```

Given below is the implementation to add breakpoints in a program used for example 4.

```
C:\Users\admin\Desktop\CS_Stuff\pythoVS>python -m pdb exppdb.py
> c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py(1)<module>()
-> def multiply(a, b):
(Pdb) break exppdb.py:6
Breakpoint 1 at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:6
(Pdb) break exppdb.py:7
Breakpoint 2 at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:7
(Pdb) break
Num Type      Disp Enb   Where
1  breakpoint keep yes    at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:6
2  breakpoint keep yes    at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:7
(Pdb) █
```

Adding breakpoints

Managing Breakpoints

After adding breakpoints with the help of numbers assigned to them, we can manage the breakpoints using the **enable** and **disable** and **remove** command. **disable** tells the debugger not to stop when that breakpoint is reached, while **enable** turns on the disabled breakpoints.

Given below is the implementation to manage breakpoints using Example 4.




```
Disabled breakpoint 2 at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:7
(Pdb) enable 1
Enabled breakpoint 1 at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:6
(Pdb) break

Num Type      Disp Enb      Where
1  breakpoint keep yes    at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:6
2  breakpoint keep no     at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:7
3  breakpoint keep no     at c:\users\admin\desktop\cs_stuff\pythovs\exppdb.py:8
(Pdb) █
```

Manage_breakpoints



Like 6

[< Previous](#)

Bin Size in Matplotlib Histogram

[Next >](#)

**Multiple Density Plots with
Pandas in Python**

RECOMMENDED ARTICLES

Page : [1](#) [2](#) [3](#)

01 Debugging Python code using
breakpoint() and pdb
22, Feb 19


05 Python | Merge Python key values to
list
31, Jul 19

02 Debugging in Python with Pdb
16, Mar 21

06 Reading Python File-Like Objects
from C | Python
06, Jun 19

03 Working with the Python Debugger
15, Jun 20

07 Python | Add Logging to a Python
Script
11, Jun 19

 **04** Important differences between
Python 2.x and Python 3.x with

Python | Add Logging to Python



Article Contributed By :



tejalkadam18m
@tejalkadam18m

Vote for difficulty

Current difficulty : [Medium](#)

[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)

Improved By : [tuhinmi74jn](#)

Article Tags : [Picked](#), [python-modules](#), [Python](#)

Practice Tags : [python](#)

[Improve Article](#)[Report Issue](#)

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

GeeksforGeeks



A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305



feedback@geeksforgeeks.org



[About Us](#)[Careers](#)[In Media](#)[Contact Us](#)[Privacy Policy](#)[Copyright Policy](#)[Algorithms](#)[Data Structures](#)[SDE Cheat Sheet](#)[Machine learning](#)[CS Subjects](#)[Video Tutorials](#)[Courses](#)

News

[Top News](#)[Technology](#)[Work & Career](#)[Business](#)[Finance](#)[Lifestyle](#)[Knowledge](#)

Languages

[Python](#)[Java](#)[CPP](#)[Golang](#)[C#](#)[SQL](#)[Kotlin](#)

Web Development

[Web Tutorials](#)[Django Tutorial](#)[HTML](#)[JavaScript](#)[Bootstrap](#)[ReactJS](#)[NodeJS](#)

Contribute

[Write an Article](#)[Improve an Article](#)[Pick Topics to Write](#)[Write Interview Experience](#)[Internships](#)[Video Internship](#)