03-Python Debugger (pdb)

July 11, 2018

1 Python Debugger

You've probably used a variety of print statements to try to find errors in your code. A better way of doing this is by using Python's built-in debugger module (pdb). The pdb module implements an interactive debugging environment for Python programs. It includes features to let you pause your program, look at the values of variables, and watch program execution step-by-step, so you can understand what your program actually does and find bugs in the logic.

This is a bit difficult to show since it requires creating an error on purpose, but hopefully this simple example illustrates the power of the pdb module. *Note: Keep in mind it would be pretty unusual to use pdb in an iPython Notebook setting.*

Here we will create an error on purpose, trying to add a list to an integer

```
In [2]: x = [1,3,4]
        y = 2
        z = 3

        result = y + z
        print(result)
        result2 = y+x
        print(result2)
```

```
5
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-2-e7e4af986cb2> in <module>()
          5 result = y + z
          6 print(result)
    ----> 7 result2 = y+x
          8 print(result2)
        TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

Hmmm, looks like we get an error! Let's implement a set_trace() using the pdb module. This will allow us to basically pause the code at the point of the trace and check if anything is wrong.

```
In [4]: import pdb

        x = [1,3,4]
        y = 2
        z = 3

        result = y + z
        print(result)

        # Set a trace using Python Debugger
        pdb.set_trace()

        result2 = y+x
        print(result2)
```

```
5
--Return--
> <ipython-input-4-6c36e8161fda>(11)<module>()->None
-> pdb.set_trace()
(Pdb) y+x
*** TypeError: unsupported operand type(s) for +: 'int' and 'list'
(Pdb) x
[1, 3, 4]
(Pdb) y
2
(Pdb) q
```

```
        ---------------------------------------------------------------------------

        BdbQuit                                   Traceback (most recent call last)

        <ipython-input-4-6c36e8161fda> in <module>()
          9
         10 # Set a trace using Python Debugger
    ---> 11 pdb.set_trace()
         12
         13 result2 = y+x
    ~\Anaconda3\lib\bdb.py in trace_dispatch(self, frame, event, arg)
     50             return self.dispatch_call(frame, arg)
     51         if event == 'return':
---> 52             return self.dispatch_return(frame, arg)
     53         if event == 'exception':
     54             return self.dispatch_exception(frame, arg)


    ~\Anaconda3\lib\bdb.py in dispatch_return(self, frame, arg)
     94             finally:
     95                 self.frame_returning = None
---> 96             if self.quitting: raise BdbQuit
     97             # The user issued a 'next' or 'until' command.
     98             if self.stopframe is frame and self.stoplineno != -1:


    BdbQuit:
```

Great! Now we could check what the various variables were and check for errors. You can use 'q' to quit the debugger. For more information on general debugging techniques and more methods, check out the official documentation: https://docs.python.org/3/library/pdb.html