Introduction to debugging in Python

(+ CodeFights!)

@lpmayos

PyladiesBCN

February 25, 2015

www.meetup.com/PyLadies-BCN/

Why should we learn to debug?

Is this code snippet similar to your tried-and-true debugging techniques? Yeah, that used to be me too.

```
1 def make_pie(self, ingredients):
2     print '******WHAT IS GOING ON HERE******'
3     print ingredients
4     self.oven.preheat()
5     print self.oven.temperature
```

Yeah, the print command works, but... What if you don't know where to look? What if you are facing a huge piece of code that was written by a drunken elephant riding a tricyce? There is a better way!

Logging

If you do ever litter your code with print statements stop now. Use logging.debug instead. You'll be able to reuse that later, disable it altogether and so on ...

Take a look at the logging module

`https://docs.python.org/2/library/logging.html`

Tracing

The trace module allows you to trace program execution, generate annotated statement coverage listings, print caller/callee relationships and list functions executed during a program run. It can be used in another program or from the command line.

Take a look at the trace module

`https://docs.python.org/2/library/trace.html`

i.e.

```
1 $ python -m trace --count -C . somefile.py ...
```

Debugging

- **pdb module**: defines an interactive source code debugger for Python programs (`https://docs.python.org/2/library/pdb.html`)
- **ipdb module**: exports functions to access the IPython debugger, which features tab completion, syntax highlighting, better tracebacks, better introspection with the same interface as the pdb module (`https://pypi.python.org/pypi/ipdb`)

Getting started with *ipdb.set_trace()*


```python
# @Input1: an integer
# @Input2: an integer
# @Input3: an integer
# @Output: maximum among
    @Input1, @Input2 and
    @Input3
import ipdb

def maxOfThree(a, b, c):
    ipdb.set_trace()
    if a > b:
        if a > c:
            return a
        return b
    if b > c:
        return b
    return c

print maxOfThree(5, 2, 7)
```

```
$ python max_of_three.py
> .../code/max_of_three.py(9)
    maxOfThree()
      8     ipdb.set_trace()
----> 9     if a > b:
     10         if a > c:

ipdb> c
2
lpmayos-macbookair:code lpmayos$
```

Take the advantages offered by ipdb with *TAB* (autocomplete)


```
1  # @Output: a sorted list of all
       non-negative numbers less
       than
2  # 30 which are divisible both by 3
        and by 4
3  import ipdb
4
5  def threeAndFour():
6      result = []
7      for counter in range(30):
8          if counter % 3 == 0 or
       counter % 4 == 0:
9              ipdb.set_trace()
10             result.append(counter)
11     return result
12
13 print threeAndFour()
```

```
1  $ python divisible_3_4.py
2  > .../code/divisible_3_4.py(10)
       threeAndFour()
3       9              ipdb.set_trace
       ()
4  ---> 10             result.append(
       counter)
5       11     return result
6
7  ipdb> result.
8  result.append    result.count
9  result.extend    result.index
10 result.insert    result.pop
11 result.remove    result.reverse
12 result.sort
13 ipdb> result.
```

Execute the next statement with ***n*** (next)


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2  > ...code/max_of_three.py(9)
       maxOfThree()
3        8     ipdb.set_trace()
4  ----> 9     if a > b:
5       10         if a > c:
6
7  ipdb> n
8  > ...code/max_of_three.py(10)
       maxOfThree()
9        9     if a > b:
10 ---> 10         if a > c:
11      11             return a
12
13 ipdb> n
14 > ...code/max_of_three.py(12)
       maxOfThree()
15      11             return a
16 ---> 12         return b
17      13     if b > c:
18
19 ipdb>
```

Repeat the last debugging command with *ENTER*


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2  > ...code/max_of_three.py(9)
       maxOfThree()
3        8     ipdb.set_trace()
4  ----> 9     if a > b:
5       10         if a > c:
6
7  ipdb> n
8  > ...code/max_of_three.py(10)
       maxOfThree()
9        9     if a > b:
10 ---> 10         if a > c:
11      11             return a
12
13 ipdb>
14 > ...code/max_of_three.py(12)
       maxOfThree()
15      11             return a
16 ---> 12         return b
17      13     if b > c:
18
19 ipdb>
```

Help! How do I quit? with *q* (quit)


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2  > ...code/max_of_three.py(9)
       maxOfThree()
3        8     ipdb.set_trace()
4  ----> 9     if a > b:
5       10         if a > c:
6
7  ipdb> q
8  Exiting Debugger.
```

Print the value of a variable with ***p*** (print)


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2  > ...code/max_of_three.py(9)
       maxOfThree()
3        8     ipdb.set_trace()
4  ----> 9     if a > b:
5       10         if a > c:
6
7  ipdb> p a
8  5
9  ipdb>
```

Turning off the (Pdb) prompt with ***c*** (continue)


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2  > ...code/max_of_three.py(9)
       maxOfThree()
3        8     ipdb.set_trace()
4  ----> 9     if a > b:
5       10         if a > c:
6
7  ipdb> c
8  2
9  lpmayos-macbookair:code lpmayos$
```

See where you are with *l* (list)


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2
3  ...
4
5  ipdb> n
6  > ...code/max_of_three.py(12)
       maxOfThree()
7      11              return a
8  ---> 12          return b
9      13      if b > c:
10
11 ipdb> l
12      7 def maxOfThree(a, b, c):
13      8     ipdb.set_trace()
14      9     if a > b:
15     10         if a > c:
16     11             return a
17 ---> 12         return b
18     13     if b > c:
19     14         return b
20     15     return c
21     16
22     17 print maxOfThree(5, 2, 7)
23
24 ipdb>
```

Step into subroutines with *s* (step into)

```
1  # @Output: a sorted list of
       all non-negative
   numbers less than
2  # 30 which are divisible
       both by 3 and by 4
3  import ipdb
4
5  def function2(param_a,
       param_b):
6      if param_a > param_b:
7          return param_a
8      else:
9          return param_b
10
11 def function(param_a,
       param_b):
12     ipdb.set_trace()
13     return function2(
       param_a, param_b)
14
15 print function(7, 5)
```

```
1  $ python subroutines.py
2  > .../code/subroutines.py(13)
       function()
3        12     ipdb.set_trace()
4  ---> 13     return function2(
       param_a, param_b)
5        14
6
7  ipdb> s
8  --Call--
9  > .../code/subroutines.py(5)
       function2()
10       4
11 ----> 5 def function2(param_a,
       param_b):
12       6     if param_a > param_b:
13
14 ipdb> n
15 > .../code/subroutines.py(6)
       function2()
16       5 def function2(param_a,
       param_b):
17 ----> 6     if param_a > param_b:
18       7         return param_a
19
20 ipdb>
```

Continue to the end of the current subroutine with *r* (return)


```
1  # @Output: a sorted list of
       all non-negative
     numbers less than
2  # 30 which are divisible
       both by 3 and by 4
3  import ipdb
4
5  def function2(param_a,
       param_b):
6      if param_a > param_b:
7          return param_a
8      else:
9          return param_b
10
11 def function(param_a,
       param_b):
12     ipdb.set_trace()
13     return function2(
       param_a, param_b)
14
15 print function(7, 5)
```

```
1  $ python subroutines.py
2  > .../code/subroutines.py(13)
       function()
3       12      ipdb.set_trace()
4  ---> 13      return function2(
       param_a, param_b)
5       14
6
7  ipdb> s
8  --Call--
9  > .../code/subroutines.py(5)
       function2()
10       4
11 ----> 5 def function2(param_a,
       param_b):
12       6      if param_a > param_b:
13
14 ipdb> r
15 --Return--
16 7
17 > .../code/subroutines.py(7)
       function2()
18       6      if param_a > param_b:
19 ----> 7          return param_a
20       8      else:
21
22 ipdb>
```

...just be a little careful!


```
1  # @Input1: an integer
2  # @Input2: an integer
3  # @Input3: an integer
4  # @Output: maximum among
       @Input1, @Input2 and
       @Input3
5  import ipdb
6
7  def maxOfThree(a, b, c):
8      ipdb.set_trace()
9      if a > b:
10         if a > c:
11             return a
12         return b
13     if b > c:
14         return b
15     return c
16
17 print maxOfThree(5, 2, 7)
```

```
1  $ python max_of_three.py
2  > .../code/max_of_three.py(9)
       maxOfThree()
3       8     ipdb.set_trace()
4  ----> 9     if a > b:
5       10        if a > c:
6
7  ipdb> b = "a new string"
8  *** The specified object '= "a new
       string"' is not a function
9  or was not found along sys.path.
10 ipdb>
```

What happens is that pdb attempts to execute the "b" command and it interprets the rest of the line as an argument to the "b" command

Sources

- Debugging in Python, 2009 post by Steve Ferg: `https://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/`
- Python debugging tools, 2013 post by ionel's codelog: `http://blog.ionelmc.ro/2013/06/05/python-debugging-tools/`
- Debugging Python Like a Boss `https://zapier.com/engineering/debugging-python-boss/`

Let's practice!

1. First solve the **exercises** I prepared for this session (based on some CodeFights problems)
   - Please download them from `https://github.com/pyladies-bcn/debugging_in_python`
2. Then let's play **CodeFights**
   - If you fail a problem, try to copy the code in a file and use **ipdb** to solve it!
   - or we can play a **Tournament** instead!

Thank you :) Now you try!!