

PYTHON DEBUGGING FUNDAMENTALS

Python Debugging Fundamentals

Saturday, October 20, 2012

PyCarolinas

UNC School of Pharmacy

Chapel Hill, NC

Chris Calloway

University of North Carolina

Department of Marine Sciences



WHAT IS DEBUGGING

**A method for isolating
program errors**



WHAT IS DEBUGGING

- **Execute program one statement at a time**
- **Inspect the state of objects bound to identifiers**
- **Lather, rinse, repeat**



Python's `pdb` module to the rescue!



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py
```



INVOKING PYTHON'S DEBUGGER

`$ python -m pdb fizzbuzz.py`

Run a module as a script



INVOKING PYTHON'S DEBUGGER

`$ python -m pdb fizzbuzz.py`

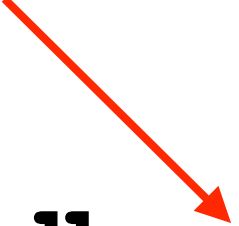
Module to run
as a script



INVOKING PYTHON'S DEBUGGER

Argument to pdb:
script to debug

```
$ python -m pdb fizzbuzz.py
```



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py  
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()  
-> ""  
(Pdb)
```



INVOKING PYTHON'S DEBUGGER

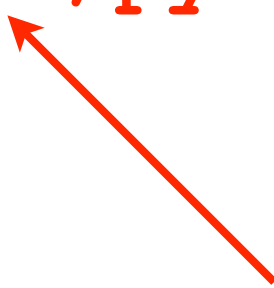
```
$ python -m pdb fizzbuzz.py
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py (7) <module> ()
```

```
-> """
```

```
(Pdb)
```

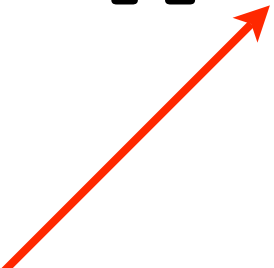
Full path to script
being debugged



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py  
> /Users/cbc/pycarolinas/fizzbuzz.py (7) <module>()  
-> ""  
(Pdb)
```

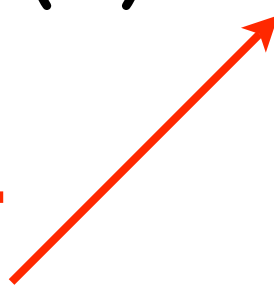
Line number of next
statement to execute



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py  
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()  
-> ""  
(Pdb)
```

Type of object
last evaluated



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py  
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()  
-> ''''''  
(Pdb)
```

**Instruction pointer to
next statement to execute**

INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py  
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()  
-> """
```

(Pdb)

Debugger prompt



PYTHON'S DEBUGGER PROMPT

(Pdb) **help**

Documented commands (type help <topic>) :

=====

EOF	cl	disable	interact	next	return	u	where
a	clear	display	j	p	retval	unalias	
alias	commands	down	jump	pp	run	undisplay	
args	condition	enable	l	print	rv	unt	
b	cont	exit	list	q	s	until	
break	continue	h	ll	quit	source	up	
bt	d	help	longlist	r	step	w	
c	debug	ignore	n	restart	tbreak	whatis	

Miscellaneous help topics:

=====

exec pdb

(Pdb)



PYTHON'S DEBUGGER PROMPT

(Pdb) **help list**

```
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. With . as argument, list 11 lines around the current line. With one argument, list 11 lines starting at that line. With two arguments, list the given range; if the second argument is less than the first, it is a count.

The current line in the current frame is indicated by "->". If an exception is being debugged, the line where the exception was originally raised or propagated is indicated by ">>", if it differs from the current line.

(Pdb)



PYTHON'S DEBUGGER PROMPT

(Pdb) **h 1**

```
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. With . as argument, list 11 lines around the current line. With one argument, list 11 lines starting at that line. With two arguments, list the given range; if the second argument is less than the first, it is a count.

The current line in the current frame is indicated by "->". If an exception is being debugged, the line where the exception was originally raised or propagated is indicated by ">>", if it differs from the current line.

(Pdb)



PYTHON'S DEBUGGER PROMPT

(Pdb) 1

2 Generate the first n Fizz Buzz answers.

3

4 Usage:

5

6 > python fizzbuzz.py n

7 -> """

8

9 import sys

10

11 def fizzbuzz(n):

12 """

(Pdb)



PYTHON'S DEBUGGER PROMPT

(Pdb) 1 6

1 """

2 Generate the first n Fizz Buzz answers.

3

4 Usage:

5

6 > python fizzbuzz.py n

7 -> """

8

9 import sys

10

11 def fizzbuzz(n):

(Pdb)



PYTHON'S DEBUGGER PROMPT

```
(Pdb) 1 .  
2     Generate the first n Fizz Buzz answers.  
3  
4     Usage:  
5  
6     > python fizzbuzz.py n  
7 -> """  
8  
9     import sys  
10  
11     def fizzbuzz(n):  
12         """  
(Pdb)
```



PYTHON'S DEBUGGER PROMPT

```
(Pdb) 3 ** (1 / 2)
```

```
1.7320508075688772
```

```
(Pdb) dir()
```

```
['__builtins__', '__file__', '__name__']
```

```
(Pdb) print(__name__)
```

```
'__main__'
```

```
(Pdb)
```



PYTHON'S DEBUGGER PROMPT

```
(Pdb) !list
```

```
<class 'list'>
```

```
(Pdb)
```



WHAT IS DEBUGGING

- Execute program **one statement at a time**
- Inspect the state of objects bound to identifiers
- Lather, rinse, repeat



SINGLE STEPPING

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(9)<module>()
```

```
-> import sys
```

(Pdb)



SINGLE STEPPING

(Pdb) 1

4 Usage:

5

6 > python fizzbuzz.py n

7 """

8

9 -> import sys

10

11 def fizzbuzz(n):

12 """

13 fizzbuzz(n) -> [first n Fizz Buzz answers]

14 """

(Pdb)



SINGLE STEPPING

```
(Pdb) dir()  
['__builtins__', '__doc__', '__file__', '__name__']  
(Pdb) !print(__doc__)
```

Generate the first n Fizz Buzz answers.

Usage:

```
> python fizzbuzz.py n
```

```
(Pdb)
```



SINGLE STEPPING

(Pdb) **s**

> /Users/cbc/pycarolinas/fizzbuzz.py(**11**)<module>()

-> def fizzbuzz(n):

(Pdb) dir()

['__builtins__', '__doc__', '__file__', '__name__', '**sys**']

(Pdb)



SINGLE STEPPING

(Pdb) 1

6 > python fizzbuzz.py n

7 """

8

9 import sys

10

11 -> def fizzbuzz(n):

12 """

13 fizzbuzz(n) -> [first n Fizz Buzz answers]

14 """

15

16 answers = []

(Pdb)



SINGLE STEPPING

(Pdb) 1

```
17         for x in range(1,n+1):
18             answer = ""
19             if not x%3:
20                 answer += "Fizz"
21             if not x%5:
22                 answer += "Buzz"
23             if not answer:
24                 answer = x
25             answers.append(answer)
26         return answers
27
```

(Pdb)



SINGLE STEPPING

(Pdb) 1



```
6      > python fizzbuzz.py n
```

```
7      """
```

```
8
```

```
9      import sys
```

```
10
```

```
11  -> def fizzbuzz(n):
```

```
12      """
```

```
13      fizzbuzz(n) -> [first n Fizz Buzz answers]
```

```
14      """
```

```
15
```

```
16      answers = []
```

(Pdb)



SINGLE STEPPING

(Pdb) **s**

> /Users/cbc/pycarolinas/fizzbuzz.py(**28**)<module>()

-> if __name__ == '__main__':

(Pdb) dir()

['__builtins__', '__doc__', '__file__',
 '__name__', '**fizzbuzz**', 'sys']

(Pdb)



SINGLE STEPPING

(Pdb) 1

```
23         if not answer:
24             answer = x
25         answers.append(answer)
26     return answers
27
28 -> if __name__ == '__main__':
29     try:
30         if len(sys.argv) != 2:
31             raise ValueError("Incorrect number of arguments")
32         answers = fizzbuzz(int(sys.argv[1]))
33         print(" ".join([str(answer) for answer in answers]))
```

(Pdb)



SINGLE STEPPING

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(29)<module>()
```

```
-> try:
```

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(30)<module>()
```

```
-> if len(sys.argv) != 2:
```

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(31)<module>()
```

```
-> raise ValueError("Incorrect number of arguments")
```

(Pdb) **s**

ValueError: Incorrect number of arguments

```
> /Users/cbc/pycarolinas/fizzbuzz.py(31)<module>()
```

```
-> raise ValueError("Incorrect number of arguments")
```

(Pdb)



SINGLE STEPPING

(Pdb) **s**

> /Users/cbc/pycarolinas/fizzbuzz.py(34)<module>()

-> except:

(Pdb) **s**

> /Users/cbc/pycarolinas/fizzbuzz.py(35)<module>()

-> print(__doc__)

(Pdb)



SINGLE STEPPING

(Pdb) **s**

Generate the first n Fizz Buzz answers.

Usage:

> python fizzbuzz.py **n**

--Return--

> /Users/cbc/pycarolinas/fizzbuzz.py (**35**) <module> () **->None**

-> print(__doc__)

(Pdb)



SINGLE STEPPING

```
(Pdb) s
```

```
--Return--
```

```
> <string>(1)<module>() ->None
```

```
(Pdb) l
```

```
[EOF]
```

```
(Pdb)
```



SINGLE STEPPING

(Pdb) **s**

```
> /opt/python330/lib/python3.3/bdb.py(409)run()
```

```
-> self.quitting = True
```

(Pdb) **s**

The program finished and will be **restarted**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()
```

```
-> """
```

(Pdb) **dir()**

```
['__builtins__', '__file__', '__name__']
```

(Pdb)



SINGLE STEPPING

(Pdb) **q**
\$



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py 100
```



INVOKING PYTHON'S DEBUGGER

```
$ python -m pdb fizzbuzz.py 100  
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()  
-> ""  
(Pdb)
```



PYTHON'S DEBUGGER PROMPT

```
(Pdb) 1
      2      Generate the first n Fizz Buzz answers.
      3
      4      Usage:
      5
      6      > python fizzbuzz.py n
      7 -> """
      8
      9      import sys
     10
     11      def fizzbuzz(n):
     12          """
(Pdb)
```



PYTHON'S DEBUGGER PROMPT

(Pdb) 1

```
13      fizzbuzz(n) -> [first n Fizz Buzz answers]
14      """
15
16      answers = []
17      for x in range(1,n+1):
18          answer = ""
19          if not x%3:
20              answer += "Fizz"
21          if not x%5:
22              answer += "Buzz"
23          if not answer:
```

(Pdb)



PYTHON'S DEBUGGER PROMPT

(Pdb) 1

```
24         answer = x
25         answers.append(answer)
26     return answers
27
28     if __name__ == '__main__':
29         try:
30             if len(sys.argv) != 2:
31                 raise ValueError("Incorrect number of arguments")
32             answers = fizzbuzz(int(sys.argv[1]))
33             print(" ".join([str(answer) for answer in answers]))
34         except:
```

(Pdb)



SETTING A BREAKPOINT

```
(Pdb) b 30
```

```
Breakpoint 1 at /Users/cbc/pycarolinas/fizzbuzz.py:30
```

```
(Pdb) 1 30
```

```
25         answers.append(answer)
26     return answers
27
28     if __name__ == '__main__':
29         try:
30 B             if len(sys.argv) != 2:
31                 raise ValueError("Incorrect number of arguments")
32                 answers = fizzbuzz(int(sys.argv[1]))
33                 print(" ".join([str(answer) for answer in answers]))
34         except:
35             print(__doc__)
```

```
(Pdb)
```



SETTING A BREAKPOINT

```
(Pdb) 1 .  
2     Generate the first n Fizz Buzz answers.  
3  
4     Usage:  
5  
6     > python fizzbuzz.py n  
7 -> """  
8  
9     import sys  
10  
11     def fizzbuzz(n):  
12         """  
(Pdb)
```



SETTING A BREAKPOINT

```
(Pdb) c
> /Users/cbc/pycarolinas/fizzbuzz.py(30)<module>()
-> if len(sys.argv) != 2:
(Pdb) l
25             answers.append(answer)
26         return answers
27
28     if __name__ == '__main__':
29         try:
30 B->             if len(sys.argv) != 2:
31                 raise ValueError("Incorrect number of arguments")
32                 answers = fizzbuzz(int(sys.argv[1]))
33                 print(" ".join([str(answer) for answer in answers]))
34         except:
35             print(__doc__)
(Pdb)
```



STEPPING INTO A FUNCTION

```
(Pdb) len(sys.argv)
```

```
2
```

```
(Pdb) s
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(32)<module>()
```

```
-> answers = fizzbuzz(int(sys.argv[1]))
```

```
(Pdb)
```



STEPPING INTO A FUNCTION

```
(Pdb) s
```

```
--Call--
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(11) fizzbuzz()
```

```
-> def fizzbuzz(n):
```

```
(Pdb) 1
```

```
6      > python fizzbuzz.py n
```

```
7      """
```

```
8
```

```
9      import sys
```

```
10
```

```
11  -> def fizzbuzz(n):
```

```
12      """
```

```
13      fizzbuzz(n) -> [first n Fizz Buzz answers]
```

```
14      """
```

```
15
```

```
16      answers = []
```

```
(Pdb)
```



STEPPING INTO A FUNCTION

```
(Pdb) p n
100
(Pdb) s
> /Users/cbc/pycarolinas/fizzbuzz.py(16)fizzbuzz()
-> answers = []
(Pdb) s
> /Users/cbc/pycarolinas/fizzbuzz.py(17)fizzbuzz()
-> for x in range(1,n+1):
(Pdb) s
> /Users/cbc/pycarolinas/fizzbuzz.py(18)fizzbuzz()
-> answer = ""
(Pdb)
```



STEPPING INTO A FUNCTION

(Pdb) 1

```
13      fizzbuzz(n) -> [first n Fizz Buzz answers]
14      """
15
16      answers = []
17      for x in range(1,n+1):
18  ->         answer = ""
19             if not x%3:
20                 answer += "Fizz"
21             if not x%5:
22                 answer += "Buzz"
23             if not answer:
```

(Pdb)



STEPPING INTO A FUNCTION

```
(Pdb) dir()
```

```
['answers', 'n', 'x']
```

```
(Pdb) where
```

```
  /opt/python330/lib/python3.3/bdb.py(405)run()
```

```
-> exec(cmd, globals, locals)
```

```
  <string>(1)<module>()
```

```
  /Users/cbc/pycarolinas/fizzbuzz.py(32)<module>()
```

```
-> answers = fizzbuzz(int(sys.argv[1]))
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(18)fizzbuzz()
```

```
-> answer = ""
```

```
(Pdb)
```



STACK FRAMES

(Pdb) **up**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(32)<module>()
```

```
-> answers = fizzbuzz(int(sys.argv[1]))
```

(Pdb) dir()

```
['__builtins__', '__doc__', '__file__',  
  '__name__', 'fizzbuzz', 'sys']
```

(Pdb) **down**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(18)fizzbuzz()
```

```
-> answer = ""
```

(Pdb) down

*** **Newest frame**

(Pdb)



STEPPING INTO A FUNCTION

(Pdb) 1

```
13      fizzbuzz(n) -> [first n Fizz Buzz answers]
14      """
15
16      answers = []
17      for x in range(1,n+1):
18  ->         answer = ""
19             if not x%3:
20                 answer += "Fizz"
21             if not x%5:
22                 answer += "Buzz"
23             if not answer:
```

(Pdb)



STEPPING INTO A FUNCTION

(Pdb) **c**

```
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17
Fizz 19 Buzz Fizz 22 23 Fizz Buzz 26 Fizz 28 29 FizzBuzz 31 32
Fizz 34 Buzz Fizz 37 38 Fizz Buzz 41 Fizz 43 44 FizzBuzz 46 47
Fizz 49 Buzz Fizz 52 53 Fizz Buzz 56 Fizz 58 59 FizzBuzz 61 62
Fizz 64 Buzz Fizz 67 68 Fizz Buzz 71 Fizz 73 74 FizzBuzz 76 77
Fizz 79 Buzz Fizz 82 83 Fizz Buzz 86 Fizz 88 89 FizzBuzz 91 92
Fizz 94 Buzz Fizz 97 98 Fizz Buzz
```

The program finished and will be restarted

```
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()
```

```
-> """
```

(Pdb)



LISTING BREAKPOINTS

(Pdb) **b**

Num	Type	Disp	Enb	Where
1	breakpoint	keep	yes	at /Users/cbc/pycarolinas/fizzbuzz.py:30

breakpoint already hit 1 time

(Pdb) **c**

> /Users/cbc/pycarolinas/fizzbuzz.py(30)<module>()

-> if len(sys.argv) != 2:

(Pdb) **s**

> /Users/cbc/pycarolinas/fizzbuzz.py(32)<module>()

-> answers = **fizzbuzz**(int(sys.argv[1]))

(Pdb)



STEPPING OVER A FUNCTION

(Pdb) **n**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(33)<module>()
```

```
-> print(" ".join([str(answer) for answer in answers]))
```

(Pdb) **pp** answers

```
[1,  
 2,  
 'Fizz',
```

```
...
```

```
98,  
 'Fizz',  
 'Buzz']
```

(Pdb)



STEPPING OVER A FUNCTION

(Pdb) **c**

```
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17
Fizz 19 Buzz Fizz 22 23 Fizz Buzz 26 Fizz 28 29 FizzBuzz 31 32
Fizz 34 Buzz Fizz 37 38 Fizz Buzz 41 Fizz 43 44 FizzBuzz 46 47
Fizz 49 Buzz Fizz 52 53 Fizz Buzz 56 Fizz 58 59 FizzBuzz 61 62
Fizz 64 Buzz Fizz 67 68 Fizz Buzz 71 Fizz 73 74 FizzBuzz 76 77
Fizz 79 Buzz Fizz 82 83 Fizz Buzz 86 Fizz 88 89 FizzBuzz 91 92
Fizz 94 Buzz Fizz 97 98 Fizz Buzz
```

The program finished and will be restarted

```
> /Users/cbc/pycarolinas/fizzbuzz.py(7)<module>()
```

```
-> """
```

(Pdb)



STEPPING OVER A FUNCTION

(Pdb) **c**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(30)<module>()
```

```
-> if len(sys.argv) != 2:
```

(Pdb) **b**

Num	Type	Disp	Enb	Where
-----	------	------	-----	-------

1	breakpoint	keep	yes	at /Users/cbc/pycarolinas/fizzbuzz.py:30
---	------------	------	-----	--

breakpoint already hit 3 times

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(32)<module>()
```

```
-> answers = fizzbuzz(int(sys.argv[1]))
```

(Pdb)



STEPPING OVER A FUNCTION

```
(Pdb) s
```

```
--Call--
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(11)fizzbuzz()
```

```
-> def fizzbuzz(n):
```

```
(Pdb) s
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(16)fizzbuzz()
```

```
-> answers = []
```

```
(Pdb) s
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(17)fizzbuzz()
```

```
-> for x in range(1,n+1):
```

```
(Pdb) s
```

```
> /Users/cbc/pycarolinas/fizzbuzz.py(18)fizzbuzz()
```

```
-> answer = ""
```

```
(Pdb)
```



STEPPING OVER A FUNCTION

(Pdb) **r**

--Return--

```
> /Users/cbc/pycarolinas/fizzbuzz.py(26)fizzbuzz()->  
    [1, 2, 'Fizz', 4, 'Buzz', 'Fizz', ...]
```

-> return answers

(Pdb)



STEPPING OVER A LIST COMPREHENSION

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(33)<module>()  
-> print(" ".join([str(answer) for answer in answers]))
```

(Pdb) **s**

--Call--

```
> /Users/cbc/pycarolinas/fizzbuzz.py(33)<listcomp>()  
-> print(" ".join([str(answer) for answer in answers]))
```

(Pdb) **s**

```
> /Users/cbc/pycarolinas/fizzbuzz.py(33)<listcomp>()  
-> print(" ".join([str(answer) for answer in answers]))
```

(Pdb)



STEPPING OVER A LIST COMPREHENSION

(Pdb) **r**

--Return--

> /Users/cbc/pycarolinas/fizzbuzz.py(**33**)<listcomp>()->

['1', '2', 'Fizz', '4', 'Buzz', 'Fizz', ...]

-> print(" ".join([str(answer) for answer in answers]))

(Pdb)



CLEAR BREAKPOINTS

```
(Pdb) h clear
```

```
cl(ear) filename:lineno
```

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

With a space separated list of breakpoint numbers, clear those breakpoints. Without argument, clear all breaks (but first ask confirmation). With a filename:lineno argument, clear all breaks at that line in that file.

```
(Pdb)
```



DISABLE BREAKPOINTS

(Pdb) h **disable**

`disable bnumber [bnumber ...]`

Disables the breakpoints given as a space separated list of breakpoint numbers. Disabling a breakpoint means it cannot cause the program to stop execution, but unlike clearing a breakpoint, it remains in the list of breakpoints and can be (re-)enabled.

(Pdb)



ENABLE BREAKPOINTS

(Pdb) h **enable**

`enable bnumber [bnumber ...]`

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

(Pdb)



MONITOR OBJECTS FOR CHANGES

```
(Pdb) h display  
display [expression]
```

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

Without expression, list all display expressions for the current frame.

```
(Pdb)
```



CONDITIONAL BREAKPOINTS

(Pdb) h **condition**

`condition bpnumber [condition]`

Set a new condition for the breakpoint, an expression which must evaluate to true before the breakpoint is honored. If condition is absent, any existing condition is removed; i.e., the breakpoint is made unconditional.

(Pdb)



SKIP OVER BREAKPOINTS

(Pdb) h **ignore**

`ignore bpnumber [count]`

Set the ignore count for the given breakpoint number. If count is omitted, the ignore count is set to 0. A breakpoint becomes active when the ignore count is zero. When non-zero, the count is decremented each time the breakpoint is reached and the breakpoint is not disabled and any associated condition evaluates to true.

(Pdb)



DEBUGGER MACROS

(Pdb) help **alias**

```
alias [name [command [parameter parameter ...] ]]
```

Create an alias called 'name' that executes 'command'. The command must **not** be enclosed in quotes. Replaceable parameters can be indicated by %1, %2, and so on, while %* is replaced by all the parameters. If no command is given, the current alias for name is shown. If no name is given, all aliases are listed.



ONE TIME BREAKPOINTS

(Pdb) h **tbreak**

```
tbreak [ ([filename:]lineno | function) [, condition] ]
```

Same arguments as break, but sets a temporary breakpoint: it is automatically deleted when first hit.

(Pdb)



WHAT IS DEBUGGING

- **Execute entire program one step at the time**
- **Execute only suspect portions of program**
- **Isolate suspect portions of program**



SETTING A TRACE

```
if __name__ == '__main__':  
    try:  
        if len(sys.argv) != 2:  
            import pdb; pdb.set_trace()  
            raise ValueError("Incorrect number of arguments")  
        answers = fizzbuzz(int(sys.argv[1]))  
        print(" ".join([str(answer) for answer in answers]))  
    except:  
        print(__doc__)
```



SETTING A TRACE

```
$ python fizzbuzzNG.py  
> /Users/cbc/pycarolinas/fizzbuzzNG.py(32)<module>()  
-> raise ValueError("Incorrect number of arguments")  
(Pdb)
```



POST-MORTEM DEBUGGING

```
if __name__ == '__main__':  
    if len(sys.argv) != 2:  
        raise ValueError("Incorrect number of arguments")  
  
    answers = fizzbuzz(int(sys.argv[1]))  
    print(" ".join([str(answer) for answer in answers]))
```



POST-MORTEM DEBUGGING

```
$ python -i fizzbuzzNG2.py
```

```
Traceback (most recent call last):
```

```
  File "fizzbuzzNG2.py", line 30, in <module>
```

```
    raise ValueError("Incorrect number of arguments")
```

```
ValueError: Incorrect number of arguments
```

```
>>>
```



POST-MORTEM DEBUGGING

```
>>> import pdb; pdb.pm()  
> /Users/cbc/pycarolinas/fizzbuzzNG2.py(30)<module>()  
-> raise ValueError("Incorrect number of arguments")  
(Pdb)
```



DEBUGGER RUNNER

```
>>> import fizzbuzz
>>> import pdb
>>> pdb.run('fizzbuzz.fizzbuzz(100)')
> <string>(1)<module>()
(Pdb) s
--Call--
> /Users/cbc/pycarolinas/fizzbuzz.py(11)fizzbuzz()
-> def fizzbuzz(n):
(Pdb)
```



Questions?

cbc@chriscalloway.org

<http://drunkenpython.org/pycarolinas.zip>

<http://docs.python.org/py3k/library/pdb.html#module-pdb>

