

Debugging with `pdb.set_trace()`

Nicole Zuckerman

Clover Health

@zuckerpunch

What is PDB?



From the docs:

```
)$ python -m pdb myscript.py
```

Or:

```
[21:28:00-nicolezuckerman~/Desktop$ python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> import pdb
[>>> import mymodule
>>> pdb.run('mymodule.test()')
```

What is this
`set_trace()` that
you're doing a
whole talk about?



set_trace(), in action

```
def is_palindrome(input_str):  
    import pdb; pdb.set_trace()  
    if input_str[::-1] == input_str:  
        return True  
    return False  
  
is_palindrome('maddam')
```

But I already have
print statements.
What more do I
need?



What's so great about `set_trace()`?

1. Inspect contents of variables during real-time execution
2. Traverse call frames
3. Travel through execution order
4. Change live code during execution

Code sample time

```
# anagrammer.py
def populate_dict(input_str, dict_of_letters, increment=True):
    incrementor = 1 if increment else -1
    import pdb; pdb.set_trace()
    for letter in input_str:
        current_count = dict_of_letters.get(letter, 0)
        dict_of_letters[letter] = current_count + incrementor
    return dict_of_letters

def is_anagram(input_str1, input_str2):
    import pdb; pdb.set_trace()
    dict_of_letters = populate_dict(input_str1, {})
    final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
    if any(final_tally_dict.values()):
        return False
    return True

is_anagram('supercalifragilisticexpialidocious', 'antidisestablishmentarianism')
```


Inspect contents of
variables during execution



Inspect contents of variables during execution

```
[09:05:39-nicolezuckerman~/Desktop$ python ~/Desktop/anagrammer.py  
> /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()  
-> dict_of_letters = populate_dict(input_str1, {})  
(Pdb) █
```

Inspecting arguments of currently paused function:

```
[09:02:36-nicolezuckerman~/Desktop$ python ~/Desktop/anagrammer.py  
> /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()  
-> dict_of_letters = populate_dict(input_str1, {})  
(Pdb) input_str1  
'supercalifragilisticexpialidocious'  
(Pdb) █
```

Inspect Variables continued: print/pretty print

```
8
9 # anagrammer.py
10 def populate_dict(input_str, dict_of_letters, increment=True):
11     incrementor = 1 if increment else -1
12     import pdb; pdb.set_trace()
13 ->     for letter in input_str:
14         current_count = dict_of_letters.get(letter, 0)
15         dict_of_letters[letter] = current_count + incrementor
16     return dict_of_letters
17
18 def is_anagram(input_str1, input_str2):
```

Inspect Variables continued: print/pretty print

```
8
9 # anagrammer.py
10 def populate_dict(input_str, dict_of_letters, increment=True):
11     incrementor = 1 if increment else -1
12     import pdb; pdb.set_trace()
13 ->     for letter in input_str:
14         current_count = dict_of_letters.get(letter, 0)
15         dict_of_letters[letter] = current_count + incrementor
16     return dict_of_letters
17
18     def is_anagram(input_str1, input_str2):
(Pdb) dict_of_letters
{'o': 2, 'e': 2, 'u': 2, 'c': 3, 'f': 1, 's': 3, 't': 1, 'i': 7, 'd': 1, 'l': 3, 'r': 2, 'a': 3,
'g': 1, 'p': 2, 'x': 1}
(Pdb) p dict_of_letters
{'o': 2, 'e': 2, 'u': 2, 'c': 3, 'f': 1, 's': 3, 't': 1, 'i': 7, 'd': 1, 'l': 3, 'r': 2, 'a': 3,
'g': 1, 'p': 2, 'x': 1}
(Pdb) █
```

Inspect Variables continued: print/pretty print

```
(Pdb) pp dict_of_letters
```

```
{'a': 3,  
 'c': 3,  
 'd': 1,  
 'e': 2,  
 'f': 1,  
 'g': 1,  
 'i': 7,  
 'l': 3,  
 'o': 2,  
 'p': 2,  
 'r': 2,  
 's': 3,  
 't': 1,  
 'u': 2,  
 'x': 1}
```

Inspect Variables continued: print/pretty print

```
(Pdb) my_json_blob = {'key': 'value', 'another_key': {'nested key': 'nested value', 'another nested key': 'another nested value', 'super nested': {'wow this is deep': 'yeah man super deep'}}}
(Pdb) pp my_json_blob
{'another_key': {'another nested key': 'another nested value',
                  'nested key': 'nested value',
                  'super nested': {'wow this is deep': 'yeah man super deep'}},
 'key': 'value'}
(Pdb) █
```

Traverse frames in call stack



Traverse Call Frames

```
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
(Pdb) bt
  /Users/nicolezuckerman/Desktop/anagrammer.py(26)<module>()
-> is_anagram('supercalifragilisticexpialidocious', 'antidisestablishmentarianism')
  /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()
-> dict_of_letters = populate_dict(input_str1, {})
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
(Pdb) █
```


Traverse Call Frames

```
[(Pdb) bt
  /Users/nicolezuckerman/Desktop/anagrammer.py(26)<module>()
-> is_anagram('supercalifragilisticexpialidocious', 'antidisestablishmentarianism')
  /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()
-> dict_of_letters = populate_dict(input_str1, {})
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
[(Pdb) input_str
'supercalifragilisticexpialidocious'
[(Pdb) input_str1
*** NameError: name 'input_str1' is not defined
[(Pdb) up
> /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()
-> dict_of_letters = populate_dict(input_str1, {})
[(Pdb) input_str1
'supercalifragilisticexpialidocious'
(Pdb) █
```

Traverse Call Frames

```
15         dict_of_letters[letter] = current_count + incrementor
16     return dict_of_letters
17
18     def is_anagram(input_str1, input_str2):
19         import pdb; pdb.set_trace()
20     ->     dict_of_letters = populate_dict(input_str1, {})
21         final_tally_dict = populate_dict(input_str2, dict_of_letters, i
22         if any(final_tally_dict.values()):
23             return False
24         return True
25
(Pdb) down
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
(Pdb) █
```

Traverse Call Frames

```
> /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()
-> dict_of_letters = populate_dict(input_str1, {})
(Pdb) s
--Call--
> /Users/nicolezuckerman/Desktop/anagrammer.py(10)populate_dict()
-> def populate_dict(input_str, dict_of_letters, increment=True):
(Pdb) █
```

Traverse Call Frames

```
-> dict_of_letters = populate_dict(input_str1, {})
(Pdb) s
--Call--
> /Users/nicolezuckerman/Desktop/anagrammer.py(10)populate_dict()
-> def populate_dict(input_str, dict_of_letters, increment=True):
(Pdb) step
> /Users/nicolezuckerman/Desktop/anagrammer.py(11)populate_dict()
-> incrementor = 1 if increment else -1
(Pdb) list
6
7     # is_palindrome('maddam')
8
9     # anagrammer.py
10    def populate_dict(input_str, dict_of_letters, increment=True):
11 ->    incrementor = 1 if increment else -1
12    import pdb; pdb.set_trace()
13    for letter in input_str:
14    current_count = dict_of_letters.get(letter, 0)
```

Travel through execution



Traveling through execution

```
def is_anagram(input_str1, input_str2):
    import pdb; pdb.set_trace()
    dict_of_letters = populate_dict(input_str1, {})
    final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
    if any(final_tally_dict.values()):
        return False
    return True
```

Traveling through execution

```
def is_anagram(input_str1, input_str2):
    import pdb; pdb.set_trace()
    dict_of_letters = populate_dict(input_str1, {})
    final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
    if any(final_tally_dict.values()):
        return False
    return True
```

```
[23:27:38-nicolezuckerman~/Desktop$ python anagrammer.py
> /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()
-> dict_of_letters = populate_dict(input_str1, {})
[(Pdb) dict_of_letters
*** NameError: name 'dict_of_letters' is not defined
[(Pdb) n
> /Users/nicolezuckerman/Desktop/anagrammer.py(21)is_anagram()
-> final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
[(Pdb) dict_of_letters
{'t': 1, 'l': 3, 'r': 2, 'g': 1, 'd': 1, 'f': 1, 'e': 2, 's': 3, 'x': 1, 'o': 2, 'c': 3, 'a': 3, 'p': 2}
(Pdb)
```

Traveling through execution

```
> /Users/nicolezuckerman/Desktop/anagrammer.py(20)is_anagram()
-> dict_of_letters = populate_dict(input_str1, {})
[(Pdb) n
> /Users/nicolezuckerman/Desktop/anagrammer.py(21)is_anagram()
-> final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
[(Pdb) c
23:45:05-nicolezuckerman~/Desktop$ █
```


Where the heck am I, anyway?

```
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
(Pdb) list
8
9     # anagrammer.py
10    def populate_dict(input_str, dict_of_letters, increment=True):
11        incrementor = 1 if increment else -1
12        import pdb; pdb.set_trace()
13    ->    for letter in input_str:
14        current_count = dict_of_letters.get(letter, 0)
15        dict_of_letters[letter] = current_count + incrementor
16        return dict_of_letters
17
18    def is_anagram(input_str1, input_str2):
(Pdb) █
```

```
-> for letter in input_str:
```

```
[(Pdb) list
```

```
15
16     # anagrammer.py
17     def populate_dict(input_str, dict_of_letters, increment=True):
18         incrementor = 1 if increment else -1
19         import pdb; pdb.set_trace()
20     ->     for letter in input_str:
21             current_count = dict_of_letters.get(letter, 0)
22             dict_of_letters[letter] = current_count + incrementor
23     return dict_of_letters
24
25     def is_anagram(input_str1, input_str2):
```

```
[(Pdb) list
```

```
26     import pdb; pdb.set_trace()
27     dict_of_letters = populate_dict(input_str1, {})
28     final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
29     if any(final_tally_dict.values()):
30         return False
31     return True
32
33
34     is_anagram('supercalifragilisticexpialidocious', 'antidisestablishmentarianism')
```

```
[EOF]
```

```
(Pdb)
```

List: The Fancy Parts

```
[(Pdb) list 17, 32
17     def populate_dict(input_str, dict_of_letters, increment=True):
18         incrementor = 1 if increment else -1
19         import pdb; pdb.set_trace()
20 ->     for letter in input_str:
21         current_count = dict_of_letters.get(letter, 0)
22         dict_of_letters[letter] = current_count + incrementor
23     return dict_of_letters
24
25     def is_anagram(input_str1, input_str2):
26         import pdb; pdb.set_trace()
27         dict_of_letters = populate_dict(input_str1, {})
28         final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
29         if any(final_tally_dict.values()):
30             return False
31         return True
32
(Pdb) █
```

Change live code
during execution



Changing live code during execution

```
> /Users/nicolezuckerman/Desktop/anagrammer.py(14)populate_dict()
-> current_count = dict_of_letters.get(letter, 0)
(Pdb) letter
's'
(Pdb) letter = 'banana'
(Pdb) n
> /Users/nicolezuckerman/Desktop/anagrammer.py(15)populate_dict()
-> dict_of_letters[letter] = current_count + incrementor
(Pdb) current_count
0
(Pdb) letter
'banana'
(Pdb) n
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
(Pdb) dict_of_letters
{'banana': 1}
(Pdb) █
```

PDB-adjacent coolness: my_thing.__dict__

```
(Pdb) Foo
<class '__main__.Foo'>
(Pdb) foo = Foo()
(Pdb) foo.__dict__
{'attr1': 'some str', 'attr2': 1, 'attr3': {'some key': 'some value'}}
(Pdb)
```

PDB-adjacent coolness: dir(my_thing)

```
[(Pdb) import datetime  
[(Pdb) now = datetime.datetime.now()  
[(Pdb) pp dir(now)
```

PDB-adjacent coolness: dir(my_thing)

```
'time',  
'timestamp',  
'timetuple',  
'timetz',  
'today',  
'toordinal',  
'tzinfo',  
'tzname',  
'utcfromtimestamp',  
'utcnow',  
'utcoffset',  
'utctimetuple',  
'weekday',  
'year']  
(Pdb) █
```


Pdb gotchas



Gotchas

pdb commands versus statements

```
[12:23:35-nicolezuckerman~/Desktop$ python anagrammer.py  
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()  
-> for letter in input_str:  
[(Pdb) p = 'I am an assigned string'  
*** SyntaxError: invalid syntax  
[(Pdb) p  
*** SyntaxError: unexpected EOF while parsing  
[(Pdb) !p = 'I am an assigned string'  
[(Pdb) !p  
'I am an assigned string'  
(Pdb) █
```

Gotchas

```
(Pdb) help
```

```
Documented commands (type help <topic>):
```

```
=====
```

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

```
Miscellaneous help topics:
```

```
=====
```

```
exec  pdb
```

```
(Pdb) █
```

Gotchas

The Enter key

```
[12:25:19-nicolezuckerman~/Desktop$ python anagrammer.py
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
[(Pdb) n
> /Users/nicolezuckerman/Desktop/anagrammer.py(14)populate_dict()
-> current_count = dict_of_letters.get(letter, 0)
[(Pdb)
> /Users/nicolezuckerman/Desktop/anagrammer.py(15)populate_dict()
-> dict_of_letters[letter] = current_count + incrementor
[(Pdb)
> /Users/nicolezuckerman/Desktop/anagrammer.py(13)populate_dict()
-> for letter in input_str:
(Pdb) █
```

Bonus! Post-mortem debugging

Code sample with exception-inducing bug

```
# anagrammer.py
def populate_dict(input_str, dict_of_letters, increment=True):
    incrementor = 1 if increment else -1
    for letter in input_str:
        current_count = dict_of_letters.get(letter, 0)
        dict_of_letters[letter] = current_count + incrementor
    return dict_of_letters

def is_anagram(input_str1, input_str2):
    dict_of_letters = populate_dict({})
    final_tally_dict = populate_dict(input_str2, dict_of_letters, increment=False)
    if any(final_tally_dict.values()):
        return False
    return True
```

Post-mortem at work

```
00:30:48-nicolezuckerman~/Desktop$ python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pdb
>>> import anagrammer
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/nicolezuckerman/Desktop/anagrammer.py", line 32, in <module>
    is_anagram('supercalifragilisticexpialidocious', 'antidisestablishmentarianism')
  File "/Users/nicolezuckerman/Desktop/anagrammer.py", line 25, in is_anagram
    dict_of_letters = populate_dict({})
TypeError: populate_dict() missing 1 required positional argument: 'dict_of_letters'
>>> pdb.pm()
> /Users/nicolezuckerman/Desktop/anagrammer.py(25)is_anagram()
-> dict_of_letters = populate_dict({})
(Pdb) █
```

Breakpoints

Break!

```
01:05:30-nicolezuckerman~/Desktop$ python -m pdb anagrammer.py
> /Users/nicolezuckerman/Desktop/anagrammer.py(2)<module>()
-> def populate_dict(input_str, dict_of_letters, increment=True):
(Pdb) break 10
Breakpoint 1 at /Users/nicolezuckerman/Desktop/anagrammer.py:10
(Pdb) c
The program finished and will be restarted
> /Users/nicolezuckerman/Desktop/anagrammer.py(2)<module>()
-> def populate_dict(input_str, dict_of_letters, increment=True):
(Pdb) █
```

Break!

```
-> def populate_dict(input_str, dict_of_letters, increment=True):  
(Pdb) break anagrammer:14  
Breakpoint 2 at /Users/nicolezuckerman/Desktop/anagrammer.py:14  
(Pdb) █
```

Clear!

```
(Pdb) break anagrammer:14
Breakpoint 2 at /Users/nicolezuckerman/Desktop/anagrammer.py:14
(Pdb) clear 2
Deleted breakpoint 2 at /Users/nicolezuckerman/Desktop/anagrammer.py:14
(Pdb) clear
Clear all breaks? y
Deleted breakpoint 1 at /Users/nicolezuckerman/Desktop/anagrammer.py:10
(Pdb) █
```



A whole new world

Questions



<https://docs.python.org/2/library/pdb.html>