

Debugging Python Code at the Command Line



Douglas Starnes

Author / Entrepreneur / Speaker

linktr.ee/douglasstarnes



Overview



pdb is a command line debugger for Python applications

The debugged application needs to add a small amount of code

Commands

- Display values of variables
- Control the execution of the debugged application
- Set and clear breakpoints
- Conditional breakpoints
- Enabling and disabling breakpoints

Evaluate Python expressions in a **pdb** session



Before debugging with the pdb debugger



Import the `pdb` module (included in the Python Standard Library)



Call the `set_trace()` function in the `pdb` module



The import and `set_trace()` call are often placed on the same line



Separate multiple Python statements with a semicolon to put them on one line



PEP-8 discourages this practice, but in this case it's often overlooked



Before debugging with the pdb debugger

```
def get_investment_info(investment):  
    name = investment["name"]  
    quantity = investment["quantity"]  
    import pdb; pdb.set_trace()  
    current_price = get_current_price(name)  
    print(f"The current price of {name} is {current_price}.")
```

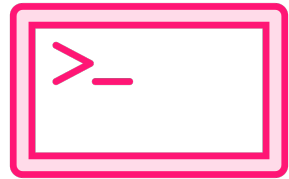


Before debugging with the pdb debugger (Python 3.7+)

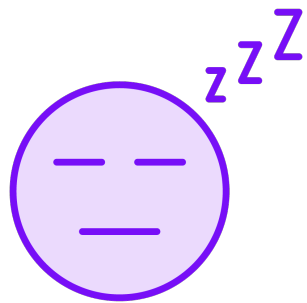
```
def get_investment_info(investment):  
    name = investment["name"]  
    quantity = investment["quantity"]  
    breakpoint() # equivalent of import pdb; pdb.set_trace()  
    current_price = get_current_price(name)  
    print(f"The current price of {name} is {current_price}.")
```



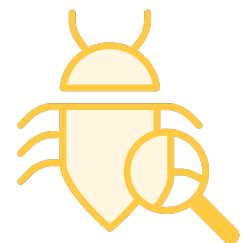
Using the pdb debugger



Invoke the application using the Python interpreter



Execution will pause at the line after the `set_trace()` call



The pdb command prompt will appear in the terminal



The pdb command prompt

```
$ python app.py  
> /root/src/ps/pdbdemo/app.py(28)get_investment_info()  
-> current_price = get_current_price(name)  
(Pdb)
```



pdb commands



help (alias **h**) – get a list of available commands



Pass the name of a command to **help** to get instructions for that command



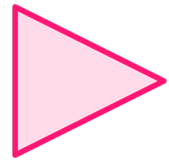
p (short for print) – display the value of a variable



pp (short for pretty print) – format and display complex values (ie. dictionaries)



pdb commands (2)



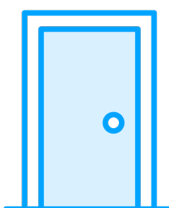
next (alias **n**) – execute until the following line of code (similar to ‘step over’)



continue (alias **c** & **cont**) – resume execution until the next breakpoint or until the application exits



quit (alias **q**) – quit the debugger and abort the application being debugged



exit – same as **quit**




The next (or n) command

```
def get_investment_info(investment):  
    name = investment["name"]  
    quantity = investment["quantity"]  
    import pdb; pdb.set_trace()  
→ current_price = get_current_price(name)  
    print(f"The current price of {name} is {current_price}.")
```



The next (or n) command

```
def get_investment_info(investment):  
    name = investment["name"]  
    quantity = investment["quantity"]  
    import pdb; pdb.set_trace()  
    current_price = get_current_price(name)   
    print(f"The current price of {name} is {current_price}.")
```

(Pdb) next



The next (or n) command

```
def get_investment_info(investment):  
    name = investment["name"]  
    quantity = investment["quantity"]  
    import pdb; pdb.set_trace()  
    current_price = get_current_price(name)  
    → print(f"The current price of {name} is {current_price}.")
```



Commands for managing breakpoints



break (alias **b**)



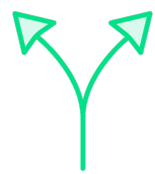
If no argument is included list all breakpoints



To set a breakpoint, provide a line number or function as the first argument



The line number or function can be preceded with a filename



The second argument is a conditional; the breakpoint is ignored if the expression evaluates to **False**



Setting breakpoints

```
$ python app.py  
> /root/src/ps/pdbdemo/app.py(28)get_investment_info()  
-> current_price = get_current_price(name)  
(Pdb)
```



Setting breakpoints

```
$ python app.py  
> /root/src/ps/pdbdemo/app.py(28)get_investment_info()  
-> current_price = get_current_price(name)  
(Pdb) break 30
```



Setting breakpoints

```
$ python app.py  
> /root/src/ps/pdbdemo/app.py(28)get_investment_info()  
-> current_price = get_current_price(name)  
(Pdb) break 30  
Breakpoint 1 at /root/src/ps/pdbdemo/app3.py:30  
(Pdb)
```



Setting breakpoints

```
$ python app.py  
> /root/src/ps/pdbdemo/app.py(28)get_investment_info()  
-> current_price = get_current_price(name)  
(Pdb) break 30  
Breakpoint 1 at /root/src/ps/pdbdemo/app3.py:30  
(Pdb) break
```

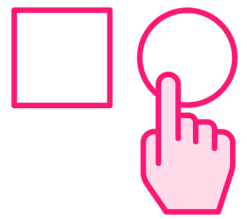


Setting breakpoints

```
$ python app.py
> /root/src/ps/pdbdemo/app.py(28)get_investment_info()
-> current_price = get_current_price(name)
(Pdb) break 30
Breakpoint 1 at /root/src/ps/pdbdemo/app3.py:30
(Pdb) break
Num Type          Disp Enb      Where
1  breakpoint    keep yes    at /root/src/ps/pdbdemo/app3.py:30
(Pdb)
```



Commands for managing breakpoints (2)



`clear` (alias `cl`) – remove one or more breakpoints



If no argument is included remove all breakpoints



To remove a specific breakpoint, provide a line number or breakpoint number as the first argument



The line number can be preceded with a filename



Summary



pdb is a command line debugger for Python included in the standard library

Add a single line of code calling a function to start a debugger session

Interact with the debugger through commands

- Inspect the state of the application
- Control application execution
- Manage breakpoints

Use Python expressions to modify the state of the debugged application

