README.md

Python-Debugging

Contains the code for the Python Debugging with PDB tutorial on Python Debugging With Pdb.

Sections

1. Getting Started: Printing a Variable's Value
2. Printing Expressions
3. Stepping Through Code
   i. Listing Source Code
4. Using Breakpoints
5. Continuing Execution
6. Displaying Expressions
7. Python Caller ID
8. Essential pdb Commands
9. Python Debugging With pdb: Conclusion

Notes

1. start debugger with

   i. `import pdb; pdb.set_trace`

   ii. `breakpoint()`

      a. `breakpoint()` is **more preferable** since you can `set PYTHONBREAKPOINT=0` and completely disable debugging

   iii. `python -m pdb app.py arg1 arg2`

2. press `q` to quit

3. example_1

   i. output:

   ```
   > j:\education\code\python\python-debugging\python_debug\python_debug.py(17)example_1()
   -> print(f'path = {filename}')
   (Pdb) p filename
   'python_debug.py'
   ```

   ii. `>` starts the 1st line and tells you which source file you're in. After the filename, there is the current line number in parentheses.

   iii. Next is the name of the function. In this example, since we're not paused inside a function and at module level, we see `<module>()`.

   iv. `->` starts the 2nd line and is the current source line where Python is paused. This line hasn't been executed yet. In this example, this is line 24 in example_1.py, from the `>` line above.

   v. `(Pdb)` is pdb's prompt. It's waiting for a command

4. example_2

   i. we can print expressions using the `p` command

      a. `ll` is **longlist** and prints the function source code

```
(Pdb) ll
    28     def get_path(filename):
    29         """
    30         Return the path of the file
    31         args:
    32             filename (str): name of the file
    33         returns:
    34             head (str): path to the file
    35         """
    36         head, tail = os.path.split(filename)
    37         breakpoint()
    38  ->     return head
    ```
```

b. print multiple expressions using `,`

```
(Pdb) p head, tail
('', 'python_debug.py')```
```

c. concatenate strings and expressions using `+`

```
(Pdb) p 'filename' + filename
'filenamepython_debug.py'
```

d. use `get_attr` to view attributes such as `__doc__`

```
(Pdb) p getattr(get_path, '__doc__')
'\n    Return the path of the file\n    args:\n        filename (str): name of the file\n    returns:\n        head
```

e. perform a short expression

```
(Pdb) pp [os.path.split(p)[1] for p in os.path.sys.path]
['python_debug', 'python38.zip', 'DLLs', 'lib', 'Python38', 'site-packages']
```

ii. the `pp` command can be used to pretty print expressions

5. example_3

i. output:

```
PS J:\Education\Code\Python\Python-Debugging\python_debug> python python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(46)example_3()
-> filename_path = get_path(filename)
(Pdb) n
> j:\education\code\python\python-debugging\python_debug\python_debug.py(47)example_3()
-> print(f'path = {filename_path}')
```

```
PS J:\Education\Code\Python\Python-Debugging\python_debug> python python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(46)example_3()
-> filename_path = get_path(filename)
(Pdb) s
--Call--
> j:\education\code\python\python-debugging\python_debug\python_debug.py(28)get_path()
-> def get_path(filename):
(Pdb) n
> j:\education\code\python\python-debugging\python_debug\python_debug.py(36)get_path()
-> head = os.path.split(filename)[0]
(Pdb)
> j:\education\code\python\python-debugging\python_debug\python_debug.py(37)get_path()
-> return head
(Pdb)
--Return--
> j:\education\code\python\python-debugging\python_debug\python_debug.py(37)get_path()->''
-> return head
(Pdb)
> j:\education\code\python\python-debugging\python_debug\python_debug.py(47)example_3()
-> print(f'path = {filename_path}')
```
```

```
(Pdb)
path =
--Return--
> j:\education\code\python\python-debugging\python_debug\python_debug.py(47)example_3()->None
-> print(f'path = {filename_path}')
```

ii. `n` command

a. **next**

b. `n` command is used to stepover in local functions,

c. wont' move into other function calls

d. remains in the same function

e. Continue execution until the next line in the current function is reached or it returns.

iii. `s` command

a. **step**

b. steps into foreign function from local function

c. Execute the current line and stop at the first possible occasion (either in a function that is called or in the current function).

iv. Notes

```
The difference between n (next) and s (step) is where pdb stops.

Use n (next) to continue execution until the next line and stay within the current function, i.e. not stop in a foreign

Use s (step) to execute the current line and stop in a foreign function if one is called. Think of step as “step into”. 

Both n and s will stop execution when the end of the current function is reached and print --Return-- along with the ret

With n (next), we stopped on line 15, the next line. We “stayed local” in <module>() and “stepped over” the call to get_

With s (step), we stopped on line 6 in the function get_path() since it was called on line 14. Notice the line --Call-- 

Conveniently, pdb remembers your last command. If you’re stepping through a lot of code, you can just press Enter to rep
Note the lines --Call-- and --Return--. This is pdb letting you know why execution was stopped. n (next) and s (step) wi

Also note ->'.' at the end of the line after the first --Return--. When pdb stops at the end of a function before it ret
```

6. List code on pdb

i. using `ll` you can long list source code of current function

ii. using `l` (list) command you can list shorter code

a. `l` prints `11` lines by default around the current line until EOF

b. passing `.` to `l` like `l .` prints 11 lines or the previous listing

7. breakpoints

i. create breakpoints with `b` command

a. `b` **file-name** `:` **line-number** `expression`

b. `b` **file-name** `:` **function-name** `expression`

```
(Pdb) b util.get_path, filename.startswith('p')
(Pdb) b util:14, head.startswith('p')
```

ii. type `c` to continue till breakpoint

iii. pdb continues program execution *until* breakpoint is reached

```
-> filename_path = util.get_path(filename)
(Pdb) b util:14
Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:14
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\util.py(14)get_path()
-> return head
(Pdb)
```

iv. type `b` to print a table of all breakpoints

```
(Pdb) b
Num Type         Disp Enb   Where
1   breakpoint   keep yes   at j:\education\code\python\python-debugging\python_debug\util.py:3
        breakpoint already hit 1 time
```

v. type `enable` **Num** to enable a breakpoint

```
(Pdb) b
Num Type         Disp Enb   Where
1   breakpoint   keep yes   at j:\education\code\python\python-debugging\python_debug\util.py:3
        breakpoint already hit 1 time
(Pdb) enable 1
Enabled breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:3
```

vi. type `disable` **Num** to disable a breakpoint

```
(Pdb) b
Num Type         Disp Enb   Where
1   breakpoint   keep yes   at j:\education\code\python\python-debugging\python_debug\util.py:3
        breakpoint already hit 1 time
(Pdb) disable 1
Disabled breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:3
```

vii. using an expression means the program breaks on a line only when the expression evaluates to true

8. example_4

```
PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(61)example_4()
-> filename_path = util.get_path(filename)
(Pdb) b util:14
Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:14
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\util.py(14)get_path()
-> return head
(Pdb) p filename, head, tail
('python_debug\\python_debug.py', 'python_debug', 'python_debug.py')
(Pdb) q
```

```
PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(61)example_4()
-> filename_path = util.get_path(filename)
(Pdb) b util.get_path
Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:3
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\util.py(11)get_path()
-> if type(filename) != str:
(Pdb) p filename
'python_debug\\python_debug.py'
(Pdb) b
Num Type         Disp Enb   Where
1   breakpoint   keep yes   at j:\education\code\python\python-debugging\python_debug\util.py:3
        breakpoint already hit 1 time
(Pdb) disable 1
Disabled breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:3
(Pdb) enable 1
Enabled breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:3
(Pdb) q
```

```
PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(61)example_4()
-> filename_path = util.get_path(filename)
(Pdb) b util.get_path, filename.startswith('p')
Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:3
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\util.py(11)get_path()
-> if type(filename) != str:
```

```
(Pdb) a
filename = 'python_debug\\python_debug.py'
(Pdb) q
```

```
PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(61)example_4()
-> filename_path = util.get_path(filename)
(Pdb) b util:14, head.startswith('p')
Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\util.py:14
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\util.py(14)get_path()
-> return head
(Pdb) p head
'python_debug'
(Pdb) a
filename = 'python_debug\\python_debug.py'
(Pdb) q
Traceback (most recent call last):
File "python_debug\python_debug.py", line 64, in <module>
    example_4()
File "python_debug\python_debug.py", line 61, in example_4
    filename_path = util.get_path(filename)
File "J:\Education\Code\Python\Python-Debugging\python_debug\util.py", line 14, in get_path
    return head
File "J:\Education\Code\Python\Python-Debugging\python_debug\util.py", line 14, in get_path
    return head
File "C:\Program Files\Python37\lib\bdb.py", line 88, in trace_dispatch
    return self.dispatch_line(frame)
File "C:\Program Files\Python37\lib\bdb.py", line 113, in dispatch_line
    if self.quitting: raise BdbQuit
bdb.BdbQuit
```

9. `unt`
    i. until command
    ii. `unt` **line-number**
    iii. `unt` command moves to a line with a higher number
    iv. if no line number is specified then it moves to the very next greater line, stepping over other lines
    v. if a line number is specified then it behaves like `s` and moves to the next line with greater value than **line-number**
10. example_5

```
PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
> j:\education\code\python\python-debugging\python_debug\python_debug.py(73)get_path_fname()
-> if type(fname) != str:
(Pdb) ll
64     def get_path_fname(fname):
65         """
66         Return the path of the file
67         args:
68             fname (str): name of the file
69         returns:
70             head (str): path to the file
71         """
72         breakpoint()
73  ->     if type(fname) != str:
74             raise TypeError
75         head, tail = os.path.split(fname)  # pylint: disable=unused-variable
76         for char in tail:
77             pass
78         return head
(Pdb) unt
> j:\education\code\python\python-debugging\python_debug\python_debug.py(75)get_path_fname()
-> head, tail = os.path.split(fname)  # pylint: disable=unused-variable
(Pdb)
> j:\education\code\python\python-debugging\python_debug\python_debug.py(76)get_path_fname()
-> for char in tail:
(Pdb)
> j:\education\code\python\python-debugging\python_debug\python_debug.py(77)get_path_fname()
-> pass
(Pdb)
> j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
-> return head
(Pdb) p char, tail
```

```
('y', 'python_debug.py')
(Pdb) q
```

11. display expressions

    i. set display with `display` **expression**

    ii. unset display with `undisplay` **expression**

    iii. display automatically shows the value of an expression if it changes

    ```
    PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(74)get_path_fname()
    -> if type(fname) != str:
    (Pdb) ll
    65      def get_path_fname(fname):
    66          """
    67          Return the path of the file
    68          args:
    69              fname (str): name of the file
    70          returns:
    71              head (str): path to the file
    72          """
    73          breakpoint()
    74  ->      if type(fname) != str:
    75              raise TypeError
    76          head, tail = os.path.split(fname)  # pylint: disable=unused-variable
    77          for char in tail:
    78              pass
    79          return head
    (Pdb) b 78
    Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\python_debug.py:78
    (Pdb) c
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
    -> pass
    (Pdb) display char
    display char: 'p'
    (Pdb) c
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
    -> pass
    display char: 'y'  [old: 'p']
    (Pdb) c
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
    -> pass
    display char: 't'  [old: 'y']
    (Pdb) c
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
    -> pass
    display char: 'h'  [old: 't']
    (Pdb) c
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
    -> pass
    display char: 'o'  [old: 'h']
    (Pdb) q
    ```

    iv. you can see all expressions with `display`

    ```
    PS J:\Education\Code\Python\Python-Debugging> python python_debug\python_debug.py
    > j:\education\code\python\python-debugging\python_debug\python_debug.py(74)get_path_fname()
    -> if type(fname) != str:
    (Pdb) ll
    65      def get_path_fname(fname):
    66          """
    67          Return the path of the file
    68          args:
    69              fname (str): name of the file
    70          returns:
    71              head (str): path to the file
    72          """
    73          breakpoint()
    74  ->      if type(fname) != str:
    75              raise TypeError
    76          head, tail = os.path.split(fname)  # pylint: disable=unused-variable
    77          for char in tail:
    78              pass
    ```

```
79          return head
(Pdb) b 78
Breakpoint 1 at j:\education\code\python\python-debugging\python_debug\python_debug.py:78
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
-> pass
(Pdb) display
Currently displaying:
(Pdb) display char
display char: 'p'
(Pdb) display fname
display fname: 'python_debug\\python_debug.py'
(Pdb) display head
display head: 'python_debug'
(Pdb) display tail
display tail: 'python_debug.py'
(Pdb) c
> j:\education\code\python\python-debugging\python_debug\python_debug.py(78)get_path_fname()
-> pass
display char: 'y'  [old: 'p']
(Pdb) display
Currently displaying:
char: 'y'
fname: 'python_debug\\python_debug.py'
head: 'python_debug'
tail: 'python_debug.py'
(Pdb) q
```

12. Use the `w` command to know where you are

    i. `w` means where
    ii. use up `u` or down `d` to move around the frames
    iii. **the most recent frame is at the bottom, start there and read from the bottom up**
    iv. A stack trace is just a list of all the frames that Python has created to keep track of function calls. A frame is a data structure Python creates when a function is called and deletes when it returns. The stack is simply an ordered list of frames or function calls at any point in time. The (function call) stack grows and shrinks throughout the life of an application as functions are called and then return. When printed, this ordered list of frames, the stack, is called a stack trace
    v. Think of the current frame as the current function where pdb has stopped execution. In other words, the current frame is where your application is currently paused and is used as the "frame" of reference for pdb commands like p (print). p and other commands will use the current frame for context when needed. In the case of p, the current frame will be used for looking up and printing variable references. When pdb prints a stack trace, an arrow > indicates the current frame.
    vi. to move multiple frames specify the count variable(default 1):

        a. `u` 2

        ```
        > j:\education\code\python\python-debugging\python_debug\python_debug.py(89)get_file_info()
                -> file_path = fileutil.get_path(full_fname)
                (Pdb) d
                > j:\education\code\python\python-debugging\python_debug\fileutil.py(13)get_path()
                -> if type(filename) != str:
                (Pdb) u 2
                > j:\education\code\python\python-debugging\python_debug\python_debug.py(94)example_6()
                -> filename_path = get_file_info(filename)
                (Pdb) q
        ```

        b. `d` 2

    vii. example_6

```
> j:\education\code\python\python-debugging\python_debug\fileutil.py(13)get_path()
-> if type(filename) != str:
(Pdb) w
j:\education\code\python\python-debugging\python_debug\python_debug.py(97)<module>()
-> example_6()
j:\education\code\python\python-debugging\python_debug\python_debug.py(94)example_6()
-> filename_path = get_file_info(filename)
j:\education\code\python\python-debugging\python_debug\python_debug.py(89)get_file_info()
-> file_path = fileutil.get_path(full_fname)
> j:\education\code\python\python-debugging\python_debug\fileutil.py(13)get_path()
-> if type(filename) != str:
(Pdb) u
> j:\education\code\python\python-debugging\python_debug\python_debug.py(89)get_file_info()
```

```
-> file_path = fileutil.get_path(full_fname)
(Pdb) d
> j:\education\code\python\python-debugging\python_debug\fileutil.py(13)get_path()
-> if type(filename) != str:
(Pdb) u 2
> j:\education\code\python\python-debugging\python_debug\python_debug.py(94)example_6()
-> filename_path = get_file_info(filename)
(Pdb) q
```

13. '`h` is the help command

    i. use `h` **command**

    ii. example: `h w`

Quick reference

Command	Description
p	Print the value of an expression.
pp	Pretty-print the value of an expression.
n	Continue execution until the next line in the current function is reached or it returns.
s	Execute the current line and stop at the first possible occasion (either in a function that is called or in the current function).
c	Continue execution and only stop when a breakpoint is encountered.
unt	Continue execution until the line with a number greater than the current one is reached. With a line number argument, continue execution until a line with a number greater or equal to that is reached.
l	List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing.
ll	List the whole source code for the current function or frame.
b	With no arguments, list all breaks. With a line number argument, set a breakpoint at this line in the current file.
w	Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame, which determines the context of most commands.
u	Move the current frame count (default one) levels up in the stack trace (to an older frame).
d	Move the current frame count (default one) levels down in the stack trace (to a newer frame).
h	See a list of available commands.
h	Show help for a command or topic.
h pdb	Show the full pdb documentation.
q	Quit the debugger and exit.