Python Debugger (PDB)

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2020/index.html

Starting the debugger

```
pdb3 program.py arg1 arg2
```
⟵ more convenient than gdb

OR

```
python3 -m pdb program.py arg1 arg2
```

Starting the debugger

```
pdb3 program.py arg1 arg2
```
⟵ more convenient than gdb

OR

```
python3 -m pdb program.py arg1 arg2
```

Alternatives: insert the following at location where you want execution to stop

```
import pdb; pdb.set_trace()
```

For Python 3.7 onwards:

```
breakpoint()
```

Setting environment variable `PYTHONBREAKPOINT` to 0 in your environment disables debugging

```
$ PYTHONBREAKPOINT=0 python3 program.py
```

OR

```
$ export PYTHONBREAKPOINT=0
$ python3 program.py
```

pdb commands (**same as gdb**) I

Viewing code, expressions

- `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.
- `p(rint) <expr>`
  Any valid Python expression can be passed.

pdb commands (**same as gdb**) II

- `disp(lay) [expression]`
  `undisplay [expression]`

  watchpoints

  Display the value of expression *if it changed*, each time execution stops in the current frame. Without expression, list all display expressions for the current frame.
  Do not display expression any more in the current frame. Without expression, clear all display expressions for the current frame.

pdb commands (**same as gdb**) III

Breakpoints

- `b(reak) [ ([filename:]lineno | function) [,condition] ]`
- `cond(ition) 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.
- `dis(able) bpnumber`
  `en(able) bpnumber`
- `cl(ear) filename:lineno`
  `cl(ear) [bpnumber [bpnumber...]]`
- `tb(reak)` : set temporary breakpoint

pdb commands (**same as gdb**) IV

Executing statements

- `c(ontinue)`
- `n(ext)` (step over)
  Continue execution until the next line in the current function is reached or it returns.
- `s(tep)` (step into)
  Execute the current line and stop at the first possible occasion (either in a function that is called or in the current function).
- `unt(il) [lineno]` *(very useful for loops)*
  Without lineno, continue execution until the line with a number greater than the current one is reached. With lineno, continue execution until a line with a number greater or equal to that is reached. In both cases, also stop when the current frame returns.

pdb commands (**same as gdb**) V

Examining / navigating the stack

- `w(here)` OR `bt`
  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(p) [count]`
  Move the current frame count (default one) levels up in the stack trace (to an older frame).
- `d(own) [count]`
  Move the current frame count (default one) levels down in the stack trace (to a newer frame).

Misc.

- `<Enter>` : repeat last command
- `q(uit)`

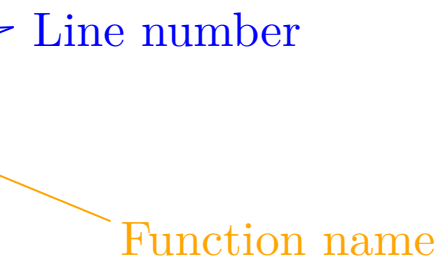
pdb commands (**different from gdb**)

- `b` (with no arguments) : list all breakpoints
- `r(eturn)` : continue execution until current function returns
- `ll` (longlist) : list the whole source code for the current function or frame
- `pp` (pretty-print) : helpful for printing variable or expression with large amount of output, e.g., lists and dictionaries
- `a` (args) : prints the argument list of current function

not run

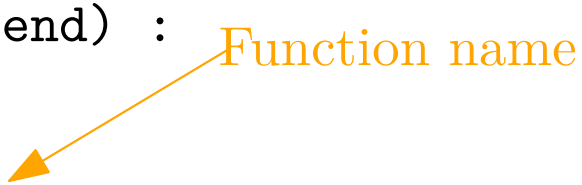
Output printed by pdb

```
(Pdb) run
Restarting quicksort.py with arguments:
        quicksort.py 4
> ./quicksort.py(1)<module>()
-> from random import *
(Pdb) n
> ./quicksort.py(2)<module>()
-> import sys
(Pdb) l
 33         if N <= 0 :
 34             sys.exit('Usage: %s <number of elements>' % sys.argv[0])
 35
 36         L = sample(range(1000000), N)
 37         start_time = time.process_time()
 38  ->     for _ in range(1000) :
 39             A = L.copy()
 40             quicksort(A, 0, N-1)
 41             for i in range(N-1) :
 42                 assert A[i] <= A[i+1]
 43         end_time = time.process_time()
(Pdb)
```



Output printed by pdb

```
(Pdb) n
> ./quicksort.py(40)<module>()
-> quicksort(A, 0, N-1)
(Pdb) s
--Call--
> ./quicksort.py(21)quicksort()
-> def quicksort(A, start, end) :
(Pdb) r
--Return--
> ./quicksort.py(25)quicksort()->None
-> quicksort(A, pivot+1, end)
(Pdb) n
> ./quicksort.py(41)<module>()
-> for i in range(N-1) :
(Pdb)
```



References

- https://realpython.com/python-debugging-pdb/