Debugging

pdb implements an interactive debugging environment for Python programs. It includes features to let you pause your program, look at the values of variables, and watch program execution step-by-step, so you can understand what your program actually does and find bugs in the logic.

Starting the Debugger

From the Command Line

In [1]:

```
def seq(n):
    for i in range(n):
        print(i)
    return

seq(5)
```

```
0
1
2
3
4
```

From Within Your Program

In [ ]:

```
import pdb

#interactive debugging
def seq(n):
    for i in range(n):
        pdb.set_trace() # breakpoint
        print(i)
    return

seq(5)


# c : continue
# q: quit
# h: help
# list
# p: print
# p locals()
# p globals()
```

```
> <ipython-input-2-81a57f73998e>(7)seq()
-> print(i)
(Pdb) print n
*** SyntaxError: Missing parentheses in call to 'print'. Did you mean print(n)?
(Pdb) p i
0
(Pdb) p i
0
(Pdb) c
0
> <ipython-input-2-81a57f73998e>(6)seq()
-> pdb.set_trace() # breakpoint
(Pdb) p i
1
(Pdb) c
1
> <ipython-input-2-81a57f73998e>(7)seq()
```

```
-> print(i)
(Pdb) p i
2
(Pdb) print i
*** SyntaxError: Missing parentheses in call to 'print'. Did you mean print(i)?
(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) print (i)
2
(Pdb) list
  2
  3   #interactive debugging
  4   def seq(n):
  5       for i in range(n):
  6           pdb.set_trace() # breakpoint
  7  ->         print(i)
  8       return
  9
 10   seq(5)
 11
 12
(Pdb) b 6
Breakpoint 1 at <ipython-input-2-81a57f73998e>:6
```

Debugger Commands

1. h(elp) [command]

Without argument, print the list of available commands. With a command as argument, print help about that command. help pdb displays the full documentation (the docstring of the pdb module). Since the command argument must be an identifier, help exec must be entered to get help on the ! command.

2. w(here)

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.

3. d(own) [count]

Move the current frame count (default one) levels down in the stack trace (to a newer frame).

4.c(ont(inue))

Continue execution, only stop when a breakpoint is encountered.

5. q(uit)

Quit from the debugger. The program being executed is aborted.

Termial/Command prompt based debugging

In [ ]:

```
Top pdb Commands
```

```
Command Key Description
Next n Execute the next line
Print p Print the value of the variable following p
Repeat Enter Repeat the last entered command
List l Show few lines above and below the current line
Step s Step into a subroutine
Return r Run until the current subroutine returns
Continue c Stop debugging the current breakpoint and continue normally
Quit q Quit pdb abruptly
```

How to invoke pdb without even modifying the script?

In [ ]:

```
python3 -m pdb sample.py
```

How to start an interactive shell once the program terminates with an error?

In [ ]:

```
python3 -i sample.py
```

Save execution trace in a log file

In [ ]:

```
python -m trace -t sample.py > execution.log
```

In [ ]:

```
bytecode is cross-platform , but is not cross-version.
```