

# Debugging 2 & useful small modules

Week 5, day 4

# pdb

- `import pdb`
- Interactive debugger
  - Stepping through code
  - Programmatically invocable
  - Introspect code

# run and step

- `pdb.run ('cmd')`
- `s(tep)` through code a line or call at a time
- Other `pdb` commands

```
>>> import pdb

>>> pdb.run ("import
myscript.py")

(Pdb) s
--Call--
> /Users/agapow/test.py(3)?()
-> print 1
```

# Pdb commands

- `s(tep)`: execute next line or call
- `n(ext)`: next line
- `r(eturn)`: complete current function
- `c(ontinue)`: run

# Pdb commands 2

- `l(ist)`: print source around current line
- `a(rgs)`: arguments of current function
- `p expr`: run expression in context of current line
- `w(here)`: where are we? Give stack of calls

# Demo: Stepping through code

```
def build_concensus (seqs):  
    assert (0 < len (seqs))  
    theAns = ''  
    for item in seqs:  
        print item  
        theAns += item  
    return theAns  
  
seq_list = ['abc', 'def', 'ghi']  
theCon = build_concensus (seq_list)  
print theCon
```

# So ...

- `s(step)`: execute current statement and move to next executable statement
- `n(ext)`: execute current statement and move to next executable statement **on same level / in current function**
- `c(ontinue)`: run until next breakpoint

# Programmatically activating debugger

- `set_trace()`: drops into debugger

```
def buggy_func():  
    # this isn't working  
    pdb.set_trace()
```

```
def buggy_func2 (arg):  
    if (arg < 0):  
        # this isn't working  
        pdb.set_trace()
```

# Programmatically activating debugger 2

- Breakpoints:  
doing `set_trace`  
within `pdb`
- `b [loc] [cond]`
- `loc` is:
  - `[filename:]line`
  - `[filename:]func`

```
(Pdb) b 10
(Pdb) b xml:10
(Pdb) b parse
(Pdb) b xml:parse
(
```

# And so ...

- Breakpoints are ways of calling the debugger at set locations or given circumstances
- `tbreak`: set a “one use” breakpoint
- `cl(ear)`: remove all breakpoints
- Help defensive programming and catching non-deterministic bugs

# Useful small modules: types

- import types
- All the basic types
- Test against `type(var)`

```
BooleanType  
ListType  
IntType  
StringType  
DictType  
ClassType  
UnicodeType  
FunctionType  
MethodType  
[...]
```

```
assert (type(v) ==  
        types.IntType)
```

# Using types

- Assertion of correct arguments
- Allows **overloading** functions (Makes up for weak polymorphism)

```
def trans (target):  
    """  
    Transforms args.  
    """  
    if not (type(target) in  
            [ListType, TupleType]):  
        target = [target]  
    for item in target:  
        # do stuff
```

# Using types 2

- Most are constructors / converters
  - e.g. `types.ListType(s)`
- Many equivalent to global names
  - e.g. `list`, `str`, `int`, `float`
- What about “like” types

# Useful small modules: textwrap

- `import textwrap`
- `fn (text, width, initial_indent, subsequent_indent, expand_tabs, replace_whitespace, fix_sentence_endings, break_long_words)`
- `textwrap.fill`: wrapped text as one string
- `textwrap.wrap`: wrapped text as a sequence of strings

# Useful small modules:

## CSV

- `import csv`
- Reads comma-separated value files
- `csv.reader (file [,options])`
  - returns an iterable reader
- `csv.writer (file, [,options])`
  - `writer.write (row)`
  - `writer.writerows (rows)`

# Using csv

```
from csv import *

inFile = open ('in.csv', 'r')
theReader = reader (inFile)
for row in theReader:
    print row

outFile = open ('out.csv', 'w')
theWriter = writer (outFile)
for i in range (4):
    theWriter.write ("%s" % [i, i*2, i*3])

# also dict based readers and writers
```

# Useful large modules: os

- `import os`
- Everything to do with operating system
- Most peculiar module
- Includes `os.path`

# Using os

```
>>> import os
>>> os.name
'posix'
>>> os.getcwd()
'/Users/agapow/Documents/Projects/NBN/'
>>> os.uname()
('Darwin',
 'lapinator.local',
 '8.7.1',
 'Darwin Kernel Version 8.7.1: Wed Jun  7
16:19:56 PDT 2006;
root:xnu-792.9.72.obj~2/RELEASE_I386',
 'i386')
```

# Using os for files

```
>>> os.linesep
'\n'
>>> os.sep
 '/'
>>> os.listdir('.')
['eg6_1.py',
 'eg6_1.pyc',
 'slides_1-1.key',
 ...
 'slides_6_1.key']
```

# Using os for files 2

- `os.mkdir (name)`: make directory
- `os.rmdir (name)`: remove directory
- `os.remove (name)`: delete file
- `os.rename (old, new)`: change name
- `os.chdir (path)`: change working directory

# Using os for files: walk

```
# iterate across file system

>>> fsiter = os.walk('.')
>>> for item in fsiter:
    print item

('.', ['Biocomputing book', 'Exercises',
'LiveWires-2.0.1', 'Slides', 'Sources and
scraps'], ['.DS_Store', 'Icon\r',
'nbn_project.tmpproj'])
[...]
```

# tuples of (curr\_dir, dirs, files)