PTN-102

Python 2.x -3.x in practice

Objectives

- Part 1: Introduction
  - Features, PVM
  - Control structures/ function syntax
  - Basic types (numeric, string)
  - Debugging
- Part 2: Sequence types
  - Sequence types (list,tuple)
  - Dict (defaultdict, Counter)
  - File operations
  - Extra control structures (with/as, comprehensions)
  - Modules
- Part 3: Regular expressions
  - Regural expression elements
  - Regexp in python
- Part 4: Beyond hello world
  - Inline documentation
  - Testing (doctest, unittest, nose)
  - Logging
  - Parallel processing
- Part 5: Advanced topics
  - Generators
  - Decorators
  - Performance tuning
  - OO in python
    - Old and new style classes
    - Constructors/destructor
    - Properties
    - Operator overload
    - Abstract class, metaclass

Why python?

- Dynamically/strongly typed
  - not that lazy as shells
- “Java like” interpreter
  - automated memory management
  - cross platform execution
  - compiled byte-code (pyc)
- Support for OO
  - OO lacks basic element, such as private for attributes/functions
  - little bit better than in Perl, but far from that you see in C++/Java
- simple constructs, lost of syntactic sugar
- enforces consistent code formatting
  - maybe a headache for someone
- easy packaging, lots of third party libs
  - Package with pkg_resources, distribute on pypi.python.org
  - Deploy via pip-python/easy_install

goo.gl/Yhvv66

Executing Python Code

- minimal syntactic checks
  - the interpreter compiles sources to byte code
  - compiled python files have `.pyc` suffix
  - speed optimization
  - timestamp based recompilation
  - can generate in-memory byte code
- the Python Virtual Machine (PVM) executes byte code
  - a “big loop that iterates through byte code instructions”
  - manages memory allocation and run time checks

Implications of PVM

- no compile time phase
  - creation/evaluation of variables/classes are made runtime
- code can be change on fly
  - no need to recompile everything
- faster than shells
  - pvm does a lot of optimizations
- faster programming
  - easy memory management
  - fast write-run-fix cycles

Execution model variations

- Cpython
- PyPy
  - python interpreter made in python
- Jython
  - JSR-223 comform JAVA bytecode generation
- IronPython
  - compiles classes to .NET CLR
- Shedskin
  - translates Python to C++
    - variables can only have a single static type
    - collections are made from one type
- Frozen binaries
  - PyInstaller, cxFreeze

Running python scripts

```
# foo.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
print “Hello world!”;
```

```
$> python foo.py
Hello world!
$> chmod +x foo.py
$> foo.py
Hello world!
```

```
$> python
>>> 2+3
5
>>> i=1
>>> type(i)
<type 'int'>
>>> dir(i)
['__abs__',   '__add__',   '__and__',   '__class__',
'__cmp__', '__coerce__'
...
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError'
..
'xrange', 'zip']
>>> __builtins__.xrange.__doc__
'xrange([start,] stop[, step]) -> xrange object
...
>>> help(xrange)
```

Very basic control structures

OLP:311

OLP:327

- if;elif;else
  - no switch/case
- while; else
  - exit with break does not execute the else block
- ternary operator
- try/except/finally

```
>>> i=1
>>> if (i==1): print "a"
... elif (i==2): print "b"
... else: print "c"
a
>>> while 1:
...     break
... else: print "Ended normally"
>>> x=1
>>> print ("True" if x==1 else "False")
True
```

```
>>> try:
...     i=1/0
... except Exception as e:
...     print e.message
...
integer division or modulo by zero
>>> try:
...  raise(MyException("Big fat error"))
except (MyException,KeyboardInterrupt) as e:
...     print e.message
...
Big fat error
```

Function syntax

OLP:395

- pass
  - represents an empty block
- *args
  - variable length argument list (tuple)
- **kvargs
  - key/value args. Represented as dict
- Default values
  - the default value object allocated at compile time -> possible mem leak
- No type enforcement in function headers
  - Do runtime checks
  - Use module typecheck (accepts,returns)
- Scope: function has local scope
  - Use of global, nonlocal
  - LEGB rule

OLP:410

```
>>> def f():pass
...
>>> f
<function f at 0xb7666a3c>
>>> def f2(a,*args,**kvargs):
    print a,args,kvargs
>>> f2(1,2,3,arg1="value")
1 (2, 3) {'arg1': 'value'}
>>> def f3(a,b=1): print a,b
>>> f3(1)
1 1
>>> f3(a=2,b=3)
2 3
>>> def f4(l=[]):
...     l.append(1);print l
>>> f4()
[1]
>>> f4()
[1, 1]
```

```
>>> def f5(v):
     if (not isinstance(v,int)):
        raise TypeError("Not int")
>>> f5("apple")
Traceback (most recent call last):
 TypeError: Not int
```

goo.gl/kO8nXu

Basic types: numeric

int, long, float, complex

- different base

    0xff, 0b111, 011 =9 !!! , (0o11 in python 3.x)

    int(“ff”,16)

- long has arbitrary length, remember any number of digits

    2**100 = 1267650600228229401496703205376L

- Truncate to int, python3 __future__ division

    1/2 = 0

    1/2.0 = 0.5

    1//2 = 0

- Overridable operator
    - inherit from basic type
    - override __xxx__ functions

goo.gl/Ose2S

Basic types: strings

- string vs. unicode
- encode,decode
- __add__,__mul__
- format, % operator
    - See OLP:179 for more
- Input from console
    - raw_input,strip
    - upper,lower

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys
import codecs
Writer=codecs.getwriter('iso8859-2')
sys.stdout=Writer(sys.stdout,'replace'
)
s=u'abcá'
print s,type(s)
```

```
abcá <type 'unicode'>
```

```
>>> s='á'; s2=u'á'; s;s2
'\xc3\xa1'
u'\xe1'
>>> s.decode('utf8')
u'\xe1'
>>> s='a'; s+='b';s; s*=2;s
'ab'
'abab'
>>> s='abc' 'def';s
>>> 'abcdef'
>>> s[0:3]
'abc'
>>>"The age of %s is %d" % ('john',99)
'The age of john is 99'
>>> '{0} or {1}'.format(1,2)
'1 or 2'
>>> inp=raw_input('Answer?:')
Answer?:ApplE
>>> inp.strip().lower()
'apple'
```

String type

```
>>> l='a:b:c'.split(':');l
['a', 'b', 'c']
>>> ";;".join(l)
'a;;b;;c'
>>> 'a' in 'abc'
True
>>> 'apple'.startswith('a')
True
>>> 'apple'.isdigit()
False
```

- split,join
- startswith, isXXX
- module unicodedata
  - normalize,category
- module StringIO
  - StringIO,cStringIO

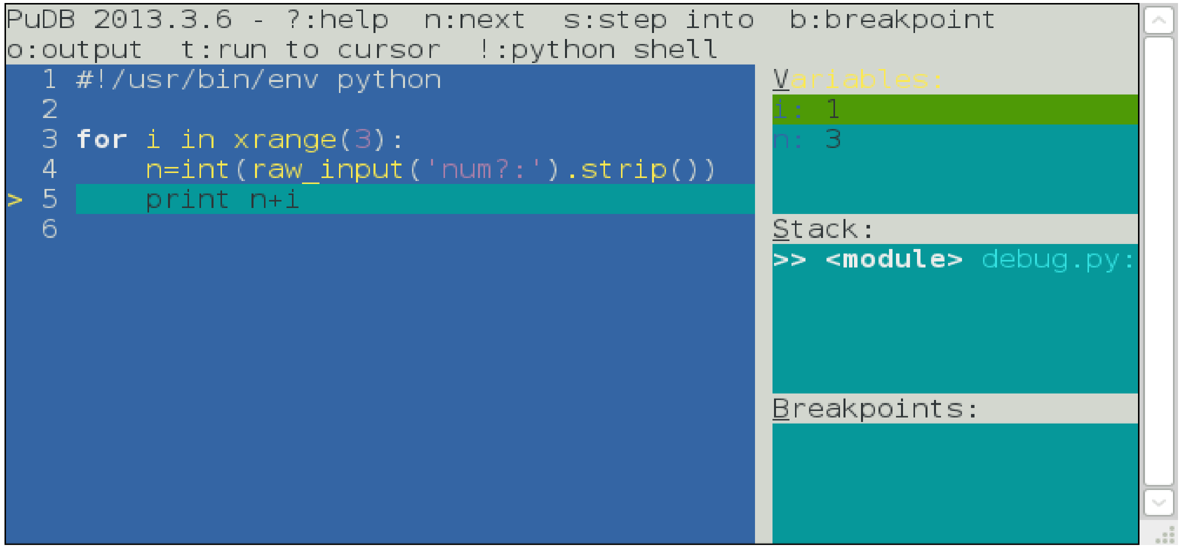
```
>>> from StringIO import StringIO
>>> s=StringIO('abc')
>>> s.seek(0,2)
>>> s.write('d')
>>> s.write('e')
>>> s.getvalue()
'abcde'
```

```
>>>   from   unicodedata   import   normalize,
category
>>> normalize('NFD',u'áé')
u'a\u0301e\u0301'
>>> for l in normalize('NFD',u'áé'):
    print "\"%s\"=\t%s" % (l, category(l))
"a"= Ll
"́"=  Mn
"e"= Ll
"́"=  Mn
```

Debugging python scripts

- pdb/ipdb
  - Built in module, similar to gdb
  - acts like a python shell
  - ipdb: uses ipython
- pudb
  - ncurses interface, great for console debugging
- Support for advanced editors
  - PyDev for Eclipse
  - Builtin IDE for Netbeans

```
$> python -mipdb /tmp/debug.py
> /tmp/debug.py(3)<module>()
----> 3 for i in xrange(3):
      4     n=int(raw_input('num?:').strip())
ipdb> n
> /tmp/debug.py(4)<module>()
      3 for i in xrange(3):
----> 4     n=int(raw_input('num?:').strip())
      5     print n+i
ipdb> i
0
```



```
PuDB 2013.3.6 - ?:help  n:next  s:step into  b:breakpoint
o:output  t:run to cursor  !:python shell
  1 #!/usr/bin/env python                           Variables:
  2                                                 i: 1
  3 for i in xrange(3):                             n: 3
  4     n=int(raw_input('num?:').strip())
> 5     print n+i                                   Stack:
  6                                                 >> <module> debug.py:
                                                    Breakpoints:
```

pydev.org

goo.gl/2rOIVp

Lab 1: Simple scripts

Note:

- Running python2.7 or python3.3 in CentOS6

```
$> yum install centos-release-SCL
$> yum install python33
$>  source /opt/rh/python33/enable
$> python3.3
Python 3.3.2 (default, Oct 30 2013, 08:01:17)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
## the same with python27
```

Sequence types

OLP:197

- Mutables vs. Inmutables
  - mutables change in place
  - mutables
    - list, array, bytearray
  - Inmutables
    - tuple, string, unicode
- default copy by reference
  - copy by value
    - use slice or constuctor
  - copy.copy/deepcopy
- slice, iteration
- access elements
  - __getitem__,__setitem__, __contains__

```
>>> l=[1,2,3]
>>> l2=l
>>> id(l), id(l2)
(3077880812L,
3077880812L)
>>> l2 is l
True
>>> l2.append(4)
>>> l
[1, 2, 3, 4]
```

```
>>> s='abc'
>>> s2=s
>>> id(s,s2)
(3077667392L, 3077667392L)
>>> s+='d'
>>> id(s),id(s2)
(3075012192L, 3077667392L)
```

```
>>> for i in "ab": print i
a
b
>>> "abcd"[1]; "abcd"[-1]
'b'
'd'
>>> "abcd"[1:3]
'bc'
>>> "abcd"[1:5:2]
'bd'
>>> 'a' in 'abc'
True
```

Other list/sequence functions

```
>>> l=[1,2]
>>> l.insert(0,0);l
[0, 1, 2]
>>> l.append(3);l
[0, 1, 2, 3]
>>> l.append(4,5);l
TypeError:  append()  takes  exactly  one
argument (2 given)
>>> l.append((4,5));l
[0, 1, 2, 3, (4, 5)]
>>> l.extend((6,7));l
[0, 1, 2, 3, (4, 5), 6, 7]
```

```
>>> l=[(1,2),(3,4),(5,-6)]
>>> sorted(l,key=operator.itemgetter(1))
[(5, -6), (1, 2), (3, 4)]
```

- insert,append,extend,pop
- sort,sorted,reverse,reversed
- range,xrange

```
>>> l=[2,3,1]
>>> l.sort();l
[1, 2, 3]
>>> l.sort(cmp=lambda x,y:cmp(y,x));l
[3, 2, 1]
>>> l.sort(reverse=True);l
[3, 2, 1]
>>> sorted(l)
[1, 2, 3]
>>> l
[3, 2, 1]
```

Dict

- key/value store, O(1) retrieval
- Facts
  - key must be a hashable type
    - inmutable
    - __hash__,__eq__ or __cmp__
  - No automatic elements
    - __setitem__ works on fly
      but __getitem__ raises KeyError
  - threading issues
    - dictiterator throws exception if the underlying dict changes during iteration

- collections.defaultdict,
- Counter (since 2.7)

```
>>> d={'a':1,'b':2}
>>> d=dict(a=1,b=2)
>>> l=[1,2,3]
>>> d[l]=1
TypeError: unhashable type: 'list'
>>> d={}; d['a']
KeyError: 'a'
>>>from collections import defaultdict
>>> d=defaultdict(str)
>>> d['a']
''
>>>from collections import Counter
>>> d2=Counter()
>>> d2['a']+=1
>>> d2['a']
1
>>> d3=Counter('aabaaccaab');d3
Counter({'a': 6, 'c': 2, 'b': 2})
```

Set, frozenset

- “Valueless dict”
    - for unique list as well
- Inmutable type since 2.7
    - frozenset
- Overloaded operators
    - Union (|)
    - Intersection (&)
    - Difference (-)

```
>>> set("abcabbbc")
set(['a', 'c', 'b'])
>>> list(set("abcabbbc"))
['a', 'c', 'b']
>>> s=set("abcd")
>>> s2=set("cef")
>>> s&s2
set(['c'])
>>> s|s2
set(['a', 'c', 'b', 'e', 'd', 'f'])
>>> s-s2
set(['a', 'b', 'd'])
>>> 'a' in s
True
```

Other sequence types/functions

- for, else
- enumerate
- map, reduce,filter
- min,max,sum
- itertools
- collections
- iterable objects

```
            in python3
>>> map(str.upper,'abc')
<map object at 0x939498c>
>>> filter(str.isupper,'aA')
<filter object at 0x9394b2c>
```

goo.gl/VPn9k

```
>>> enumerate('abc')
<enumerate object at 0xb7673e3c>
>>> list(enumerate('abc'))
[(0, 'a'), (1, 'b'), (2, 'c')]
>>> map(str.upper,['a','b'])
['A', 'B']
>>> reduce(lambda x,y:x+y,range(1,5))
10
>>> filter(str.isupper,'aAbB')
'AB'
>>> itertools.chain('ABC', 'DEF')
<itertools.chain object at 0x87958ec>
>>> list(itertools.chain('ABC', 'DEF'))
['A', 'B', 'C', 'D', 'E', 'F']
>>> d=collections.defaultdict(int)
>>> d['a']+=1;d['a']
1
>>> l=[ (1,2,3),(4,5,6),(7,8,-9) ]
>>> max(l,key=operator.itemgetter(2))
(4, 5, 6)
```

File operations

OLP:229

- open,io,codecs
- print python 2.x, 3.x
- fileinput
  - input()
  - filelineno()

```
>>> f=open('/tmp/file','r') ## for python 2.x
>>> for l in f: print l.strip()
apple
banana
>>>f=codecs.open('/tmp/file2',mode='w',encoding='utf8')
>>> print >>f , 'ű'
Traceback (most recent call last):
UnicodeDecodeError:  'ascii'  codec  can't  decode  byte
0xc5
>>> print >>f , u'ű'
>>> f.close()
```

```
>>> open('/tmp/file3',mode='w',encoding='utf8') ## python 3.x
<_io.TextIOWrapper name='/tmp/file3' mode='w' encoding='utf8'>
>>> f=open('/tmp/file3',mode='w',encoding='utf8')
>>> print('ű',file=f)
>>> f.close()
```

```
for l in fileinput.input():   # same as while(<>){print} in perl
   print "%10s:%03d %s" % (fileinput.filename(),fileinput.filelineno(),l.strip())
```

File operations – cont

- file attrs
  - os.stat
  - os.path.isdir
- file glob
  - glob.glob, iglob
  - glob.fnmatch
  - os.walk

```
>>> os.stat('/tmp/f1')
posix.stat_result(st_mode=33204,       st_ino=1218894L,
st_dev=33L,    st_nlink=1,    st_uid=500,    st_gid=500,
st_size=17L,  st_atime=1368603218,  st_mtime=1368603190,
st_ctime=1368603190)
>>> os.path.isfile('/tmp/f1')
True
>>> os.path.getatime('/tmp/f1')
1368603218.7393415
```

```
>>> glob.glob('/tmp/f*')
['/tmp/file2', '/tmp/file', '/tmp/f2', '/tmp/f1']
>>> glob.iglob('/tmp/f*')
<generator object iglob at 0xb740c1e4>
>>> for dir,dirs,files in os.walk('/tmp/dir1'):
... for file in files:
....... print os.path.join(dir,file)
/tmp/dir1/file1
/tmp/dir1/dir2/file2
```

```
>>> os.system('ls *.py 2>/dev/null')
fileinp.py   inp.py   ipython_log.py
itera.py  mod1.py
>>> os.environ['PATH']
'/usr/bin'
>>>  os.environ['VAR1']='42'
>>> os.system('echo $VAR1')
42
0
```

Communicate with external process

goo.gl/hv3jb

subprocess module: (replacement for os.system, os.popen*)

- call(args, stdin,stdout,stderr, shell)
  - forks an external process with args, waits to complete
  - std(in|out|err) is a file object (eq. subprocess.PIPE)
  - shell: if True, shell extras allowed (like file glob, redirect, environment variables)
    - Security risk combined with user input.
- Popen: exec external process with extras
  - allows non-blocking execution
  - Constructor elements: cwd, env, close_fds
  - Functions: wait, poll, communicate, kill
    - prefer communicate() instead of stdin.write/stdout read to avoid deadlock

```
>>> from subprocess import call
>>> p=call(['ls','-1'])
a1.py
b2.py
```

```
>>> from subprocess import Popen,PIPE
>>> p=Popen('ls -1 a*',shell=True, stdout=PIPE)
>>> p.stdout.readlines()
['a1.py\n']
```

Parsing command line

- getopt
  - the plain old getopt() function from libc
  - option string
    - “h” for simple option
    - “o:” for option with value
  - returns a tuple with options and remaining arguments
- argparse
  - automatically generated help
  - handles
    - -vvv (store='count')
    - -o v1 -o v2 (store='append')
  - different option types
    - str (default),int
    - file (opens the file for you)
  - since 2.7
    - use “pip install argparse” for 2.6

```
import sys
from getopt import getopt,GetoptError
try:
  (opt,args)=getopt(sys.argv[1:],
"hvo:"['help','version','output='])
except GetoptError as error:
   print "Wrong option",error;exit(-1)
opts=dict(opt);
print opt,opts,args
```

```
$> ./getopt_.py -v file1
[('-v', '')] {'-v': ''} ['file1']
$> ./geto.py -g
Wrong option option -g not recognized
```

goo.gl/rbX3DF

Parsing command line - argparse

```
import argparse
parser = argparse.ArgumentParser(description='This is an example app')
parser.add_argument('-v','--verbose',action='count',help="Make it verbose")
parser.add_argument('-o','--output',help="Write output into this file")
parser.add_argument('infiles',nargs="+",type=file,help="Input files")
args=None
try:
    args = parser.parse_args()
except Exception as e:
    parser.print_help();print e
print args
```

```
$> ./argparse_.py
usage: argparse_.py [-h] [-v] [-o OUTPUT] infiles [infiles ...]
argparse_.py: error: too few arguments
$> ./argparse_.py -vv file1
Namespace(infiles=[<open file 'file1', mode 'r' at 0xb7653cd8>],
output=None, verbose=2)
$> ./argparse_.py -vv wrong_filename
usage: argparse_.py [-h] [-v] [-o OUTPUT] infiles [infiles ...]
This is an example app
positional arguments:
  infiles               Input files
optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Make it verbose
  -o OUTPUT, --output OUTPUT   Write output into this file
[Errno 2] No such file or directory: 'wrong_filename'
```

goo.gl/SX187Y

Additional control structures – with/as

- __enter__
  - Enter the context
- __exit__
  - Exit the context (maybe because of an exception)

```
class Trace(object):
    def __init__(self,blockname):
        self.blockname=blockname

    def __enter__(self):
        print self.blockname," started"

    def __exit__(self, exc_type, exc_value, exc_tb):
        if exc_type is None:
            print self.blockname, " exited normally"
        else:
            print "Error in block ",
               self.blockname, exc_value.message
        return True # Exception is reraised if False
```

```
with Trace("block1") as blk: pass
print  "END"
with Trace("block 2") as blk:
   raise TypeError("Something bad!")
print "END"
------------
block1  started
block1  exited normally
END
block 2  started
Error in block  block 2 Something bad!
END
```

OLP:851

goo.gl/f3TQk

Additional control structures – comprehensions

- Somewhat like map, but PVM may do some optimizations

```
>>> [x.upper() for x in 'abc']                  # same as map( str.upper(),'abc')
['A', 'B', 'C']
>>> (x.upper() for x in 'abc')                      # creates a generator object
<generator object <genexpr> at 0x8c61b1c>
>>> {x.upper() for x in 'abc'}                      # 2.7 and above
{'A', 'C', 'B'}
>>> dict((x,1) for x in 'abc')                       # if inside a function call
{'a': 1, 'c': 1, 'b': 1}                              produce a generator by default
>>> d={ x:1 for x in 'abc' }                        # dict comprehension
{'a': 1, 'c': 1, 'b': 1}
>>> [ x+"a" for x in 'abc' if x<'c' ]               # optional if
['aa', 'ba']
>>> [ x+"a" for x in (x.upper() for x in 'abc') if x<'c' ]  # nested generator
['Aa', 'Ba', 'Ca']
```

goo.gl/ulLih

OLP:351

```
$> python -mtimeit  'map(str.upper,filter(lambda x:x<"c","abcd"*100))'
10000 loops, best of 3: 154 usec per loop
$> python -mtimeit '[x.upper() for x in "abcd"*100 if x<"c"]'
10000 loops, best of 3: 74.1 usec per loop
```

Modules/packages

- Modules
  - just simple .py files
- Packages
  - directories with __init__.py
    - import subpackages
    - __all__

```
mod1.py
"""This is a sample module"""
def f1(*args):
    print args
```

```
>>> import mod1
>>> dir(mod1)
['__builtins__',             '__doc__',
'__file__', '__name__', '__package__',
'f1']
>>> mod1.f1('hello')
('hello',)
>>> import mod2
>>> mod2.submod1.sayhello()
Hola
>>> mod2.submod2.doit()
Hello
```

```
$> find mod2/ -name "*.py"
mod2/submod1.py
mod2/__init__.py
mod2/submod2/doit.py
mod2/submod2/__init__.py
```

```
mod2/__init__.py
import submod1
import submod2
```

```
mod2/submod1.py
def sayhello(): print "Hola"
```

```
mod2/submod2/__init__.py
from doit import *
```

```
mod2/submod2/doit.py
def doit(): print "Hello"
```

Lab 2: Scripts with sequence types

Regexp elements

goo.gl/GPwe5

- Character ranges

  ., [abc],[a-z],\w,\W, \s,\d,
- Multipliers

  ?,*,+,{1},{1,3},{,3},{3,}
- Or structure

  john\.(jpg|gif|png)
- Anchors
  - ^,$,\b,\B,\A,\Z

```
>>> re.match(r'\w+','azAZ12-+').group(0)
'azAZ12'
>>> re.match(r'\w{2}','  az')!=None
False
>>> re.search(r'\w{2}','  az')!=None
True
>>> type(re.match(r'.{2}$','azAZ12-+'))
<type 'NoneType'>
>>> re.search(r'.{2}$','azAZ12-+').group(0)
'-+'
>>>                                 re.search(r'.
{2}$','ab\ncd',re.S).group(0)
'cd'
>>>                                 re.search(r'.
{2}$','ab\ncd',re.M).group(0)
'ab'
>>>                                 re.search(r'.
{2}\Z','ab\ncd',re.M).group(0)
'cd'
```

```
>>> re.search(r'\sapple\s',' apple ') is not None
True
>>> re.search(r'\sapple\s','apple ') is not None
False
>>> re.search(r'\bapple\s','apple ') is not None
True
```

Regexp in python

- re module functions
  - match,search
  - findall,finditer
  - split,sub
  - escape, compile

```
>>> re.findall(r'\w+','aa bb cc')
['aa', 'bb', 'cc']
>>> re.findall(r'(\w)(\w)','aa bb cc')
[('a', 'a'), ('b', 'b'), ('c', 'c')]
>>> re.finditer(r'\w+','aa bb cc').next()
<_sre.SRE_Match object at 0xb765b560>
>>> re.finditer(r'\w+','aa bb cc').next().group(0)
'aa'
>>> 'aa  bbb  ccc'.split(" ")
['aa', '', 'bbb', '', 'ccc']
>>> re.split(r'\s+','aa  bbb  ccc')
['aa', 'bbb', 'ccc']
>>> re.sub(r'[ab]','x','apple peach banana')
'xpple pexch xxnxnx'
>>> re.sub(r'[ab]',lambda x:x.group(0).upper(),'apple peach banana')
'Apple peAch BAnAnA'
```

```
>>> r=re.compile(r'a\w+e')
>>> r.search('apple pearl')
 is not None
True
>>> re.escape('100$')
'100\\$'
```

Regexp in python - cont

- re flags
  - re.I
  - re.S, re.M
  - re.X
  - re.L,re.U

```
>>> re.match(r'^[a-z]','Apple') != None
False
>>> re.match(r'^[a-z]','Apple',re.I) != None
True
>>> re.match(r'<a>(.*)</a>','<a>\napple\n</a>') != None
False
>>> re.match(r'<a>(.*)</a>','<a>\napple\n</a>',re.S) != None
True
>>> re.match(r'^(\w+) (\w+)$','aa bb') != None
True
>>> re.match(r'^(\w+) (\w+)$','aa bb',re.X) != None
False
>>> re.match(r'^(\w+)\s(\w+)$','aa bb',re.X) != None
True
>>> re.match(r'\w+','abcáé').group(0)
'abc'
>>> re.match(r'\w+','abcáé',re.U).group(0)
'abc\xe1\xe9'
```

Regexp extra

- Greediness
- Back-referencing
- Capture groups
  - named groups
  - embedded groups

```
>>> re.match(r'^(.*)(.*)(.*)$','abcd').groups()
('abcd', '', '')
>>> re.match(r'^(.*?)(.*?)(.*?)$','abcd').groups()
('', '', 'abcd')
>>> re.match(r'^(.+?)(.*?)(.+?)$','abcd').groups()
('a', '', 'bcd')
```

```
>>> re.match(r'(((\w)\w)\w)','abc').groups()
('abc', 'ab', 'a')
>>> re.match(r'^(\w)(\w).*\2\1','abccba') != None
True
>>> re.match(r'^(?P<g1>\w)(?P<g2>\w).*(?P=g2)(?P=g1)','abccba') != None
True
>>> re.match(r'^(?P<g1>\w)(?P<g2>\w).*(?P=g2)(?P=g1)','abccba').groupdict()
{'g2': 'b', 'g1': 'a'}
>>> re.sub(r'^(\w+).*?(\w+)$',r'\2 \1','aa bb')
'bb aa'
>>> re.sub(r'^(?P<g1>\w+).*?(?P<g2>\w+)$',r'\g<g2> \g<g1>','aa bb')
'bb aa'
```

Lab 3: Regural expressions

Document your code – docstrings

- Multi-line string on the first line
  - for module,class, and functions
  - format can be reStructured text (with sphinx)

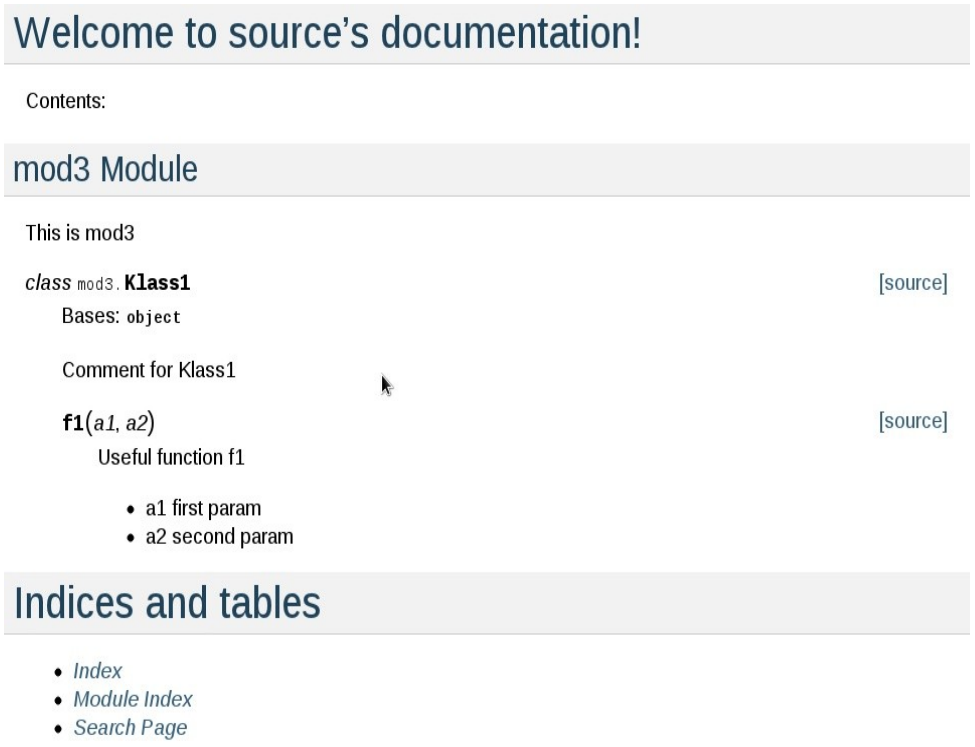
mod3.py

```
""" This is mod3"""

class Klass1(object):
    """Comment for Klass1"""

    def f1(a1,a2):
        """ Useful function f1

          - a1 first param
          - a2 second param
        """
        print a1,a2
```

```
>>> import mod3
>>> help(mod3)
Help on module mod3:
NAME
    mod3 - This is mod3
FILE
    /tmp/mod3.py
CLASSES
    __builtin__.object
        Klass1

    class Klass1(__builtin__.object)
     |  Comment for Klass1
     |
     |  Methods defined here:
     |
     |  f1(a1, a2)
     |      Useful function f1
     |      - a1 first param
     |      - a2 second param
```

Document your code - sphinx

```
$> find source -name "*.py"
source/mod3.py
$> sphinx-apidoc -F  source/ -o output
Creating file output/mod3.rst.
Creating file output/conf.py.
Creating file output/index.rst.
Creating file output/Makefile.
Creating file output/make.bat.
$> cd output
$> vi conf.py  #if customization needed
$> PYTHONPATH=../source/ make singlehtml
sphinx-build -b singlehtml -d _build/doctrees
 . _build/singlehtml
Running Sphinx v1.1.3
...
$> ls _build/singlehtml/
index.html  objects.inv  _static
```



Welcome to source's documentation!

Contents:

mod3 Module

This is mod3

class mod3.**Klass1** [source]

Bases: `object`

Comment for Klass1

f1(*a1*, *a2*) [source]

Useful function f1

- a1 first param
- a2 second param

Indices and tables

- *Index*
- *Module Index*
- *Search Page*

goo.gl/IYd1C

goo.gl/1Jbm1

Testing – doctest

- Module test with python snippets in docstrings
  - #doctest directives
    - + ELLIPSIS : ... matches any string
    - +NORMALIZE_WHITESPACE
    - +SKIP
    - +DONT_ACCEPT_BLANKLINE
    - +REPORT_UDIFF

goo.gl/DWy1o

```
"""This is mod1"""

def add(a,b):
    """This returns a+b
    >>> add(1,2)
    3
    """
    return a+b
def sub(a,b):
    """This returns a-b
    >>> sub(3,1)
    2
    """
    return 1 #Error
if __name__=="__main__":
    import doctest
    doctest.testmod()
```

```
$> python mod1.py
**********************************************
File "mod1.py", line 11, in __main__.sub
Failed example:
    sub(3,1)
Expected:
    2
Got:
    1
**********************************************
1 items had failures:
   1 of   1 in __main__.sub
***Test Failed*** 1 failures.
```

Testing – unittest

goo.gl/DbKUT

- Inherit from TestCase
- (setUp|tearDown)(Class|Module)
- separate test module
  - should implement load_tests()
- decorators
  - skip,skipIf,skipUnless
  - expectedFailure

```
"""This is mod2"""
import unittest

def add(a,b): return a+b
def div(a,b): return a/b

class MyTest(unittest.TestCase):
    @staticmethod
    def setUpClass(): print "Start testing"
    @staticmethod
    def tearDownClass(): print "Finished testing"
    def test1(self):
        self.assertEqual(add(1,2),3)
    def test2(self):
        self.assertRaises(ZeroDivisionError,div,1,0)

if __name__=="__main__":
    suite = unittest.TestLoader().
     loadTestsFromTestCase(MyTest)
    unittest.TextTestRunner(verbosity=2).run(suite)
```

```
$> python mod2.py
Start testing
test1 (__main__.MyTest) ... ok
test2 (__main__.MyTest) ... ok
Finished testing
----------------------------------
Ran 2 tests in 0.000s
OK
$> python -m unittest discover ./ -p '*.py'
Start testing ..Finished testing
Ran 2 tests in 0.000
```

Unittest + mock

- setting up the test environment is painful
  - a lot of dependencies (database,etc)
  - test setup can be complicated
    - cannot delete a file 2 times
    - do not want to drop a database just for test

```
### The Code ###
def rm(filename):
    if os.environ.has_key('LAZY'):
        os.remove(filename)
    elif os.path.isfile(filename):
        os.remove(filename)
```

```
$>  LAZY=1  python  -m  unittest  discover
./mocktest/  -p '*.py'
ERROR: test_rm (mymodule.RmTestCase)
-----------------------------------
Exception: Should not reach this!
```

```
###### Test ##########
import mock
import unittest

class RmTestCase(unittest.TestCase):
 @mock.patch('mymodule.os.path.isfile')
 @mock.patch('mymodule.os.remove')
 def test_rm(self, mock_os_remove, mock_path_isfile):

     # set up the mock
     mock_path_isfile.return_value = False
     mock_os_remove.side_effect=
      Exception("Should not reach this!")
     rm("something")
     # test that the remove call was NOT called.
     # self.assertFalse(mock_os_remove.called,
      "Should not remove file if not present.")
     # make the file 'exist'
     mock_path_isfile.return_value = True
     rm("something")
     mock_os_remove.assert_called_with("something")
```

Mock details

- Types:
  - CallableMock, NonCallableMock, PropertyMock
- Attributes, functions
  - return_value: simply return a fake value
  - side_effect:
    - Exception: it is raised when the mock is called
    - Iterable: the values from the iterable are returned
    - Callable: called with input params
  - wraps: passing the call to the wrapped object
  - spec: a class/instance or a lis of strings
    this Mock has to emulate
  - called, call_count
  - call_args, call_args_list. mock_calls
    - use the call() helper
  - assert_called_with, _once_with, assert_any_call
    - raise AssertException on call(s) with a specific set of arguments

```
### Mock can have any attribute ###
>>> m=mock.Mock()
>>> m.a
<Mock name='mock.a' id='164473100'>
>>> del(m.a)
>>> m.a
AttributeError a
```

```
### Mock with spec ###
>>> class A(object):
    __slots__=('a','b')
    def f1(*args):print args
>>>  m=mock.Mock(spec=A)
>>> m.a
<Mock name='mock.a' id='164473100'>
>>> m.c
AttributeError c
>>> m.f1(1)
<Mock name='mock.f1()' id='173215692'>
>>> m.method_calls
[call.f1(1)]
```

goo.gl/cPFdk

Testing - nose

- Combines doctest and unittest
  - recursively parses test_* modules/functions
  - A.__test__ =True to enable tests for class A

```
$> find ./ -name "*.py"
./module2.py
./package1/doit.py
./package1/__init__.py
./package1/test_with_nose.py
```

```
$> nosetests --with-doctest -v  *
package1.test_with_nose.test_that_pass
... ok
module2.test_that_fails ... FAIL
Doctest: module2.dothis ... ok
module2.test_that_fails ... FAIL
Doctest: module2.dothis ... ok
...
FAIL: module2.test_that_fails
...
TimeExpired: Time limit (0.1) exceeded
Ran 5 tests in 0.410s
FAILED (failures=2)
$>
```

package1/test_with_nose.py

```
import nose
from nose.tools import *
from doit import dothat

def setup(): pass
def teardown(): pass

@with_setup(setup,teardown)
@timed(0.1)
def test_that_pass():
    assert dothat()==1
```

module2.py

```
def dothis(*args):
    """This contains a doctest
    >>> dothis(1,2)
    1
    """
    return 1
#### This is unitest ###
import nose
from nose.tools import timed
import time

@timed(0.1)
def test_that_fails():
    time.sleep(0.2)
```

Logging in python

- Modules logging: very similar to log4j
  - Handlers: final destination for log entries
    - Such as FileHandler, See logging.handlers
  - Formatters: specifies the info to be printed from the log record
    - '%(asctime)s %(levelname)s %(message)s'
  - Logger: object assigned to a specific source, such as a module
    - logger_root: the default target for a log
    - logger.propagate: whether to pass the log also to a high level logger

`goo.gl/eZqlmO`

`goo.gl/lMMFr2`

```
import logging
import logging.config

logging.config.fileConfig("log.config")
logger=logging.getLogger(__name__)
info=logger.info
crit=logger.critical

info("Some info")
crit("Big fat error")
```

```
$> ./test_logging.py
Big fat error
$> cat 1.log
'2013-09-26 16:33:19,755 INFO Some info'
'2013-09-26 16:33:19,756 CRITICAL Big fat error'
```

Logging – config format

```
[loggers]
keys=root,logfile
[handlers]
keys=h_console,h_logfile
[formatters]
keys=f_console,f_logfile

[logger_root]
level=NOTSET
handlers=h_console,h_logfile
[logger_logfile]
level=NOTSET
handlers=h_logfile
qualname='main'

[logger_console]
level=NOTSET
handlers=h_console
```

```
[handler_h_console]
class=logging.StreamHandler
level=ERROR
formatter=f_console
args=(sys.stdout,)

[handler_h_logfile]
class=logging.handlers.RotatingFileHandler
level=DEBUG
formatter=f_logfile
args=('1.log', 'a',10000,2)

[formatter_f_console]
format=%(message)s
datefmt=
class=logging.Formatter

[formatter_f_logfile]
format=%(asctime)s %(levelname)s %(message)s
datefmt=
class=logging.Formatter
```

Parallel processing – threading

- Wraps around the “thread” module
  - dummy_threading is used, where missing
- POSIX like threading
  - Thread – executes a callable
  - Lock/Semaphore/Condition
  - synchronized block with 'with/as'
- J2EE like Queue objects
  - queue.Queue,LifoQueue,PriorityQueue

```
# like synchronized block in Java
lock=threading.RLock()
with lock:
     # atomic block of statements
```

```
class Worker(Thread):
 def __init__(self, queue):
   Thread.__init__(self)
   self.queue = queue
 def run(self):
    while 1:
      host = self.queue.get()
      url = urlopen(host)
      self.queue.task_done()
```

```
queue=Queue()
for host in urls: queue.put(host)
for i in xrange(10):
    t = Worker(queue)
    t.setDaemon(True)
    t.start()
queue.join()
```

Parallel processing - multiprocessing

- Spawning subprocesses instead using threads
  - GIL might be a limitation to use up all the cores
- Provides inter process communication primitives
  - Queue, Pipe, Lock, SHM (sharedctypes)

```
from multiprocessing import Process, Queue
def f(q):
    q.put([42, None, 'hello'])

q = Queue()
p = Process(target=f, args=(q,))
p.start()
print q.get()  # prints "[42, None, 'hello']"
p.join()
```

multip.py

```
from multiprocessing import Pool
import os
def f(x):
    print os.getpid(), x*x

p = Pool(5)
p.map(f, [1,2,3,4,5])
```

```
$> ./multip.py
17390 1
17391 4
17392 9
17393 16
17392 25
```

Advanced topics

Generator

- iterator like object to avoid temporary lists

OLP:492

```
>>> (x+1 for x in (1,2,3))
<generator object <genexpr> at 0x8ab43c4
>>> (x+1 for x in (1,2,3)).next()
2
>>> def f(): print 'Start';yield(42); print
'End'
>>> g=f()
>>> type(g)
<type 'generator'>
>>> g.next()
Start
42
>>> g.next()
End
Traceback (most recent call last):
   StopIteration
```

```
>>> def isplit(text,sep):
      i=text.find(sep,1)
      prev=0
      step=len(sep)
      while(i!=-1):
        yield(text[prev:i])
        prev=i+step
        i=text.find(sep,i+step)
      yield(text[prev:])
>>>for i in isplit('a,b,c',','):
       print i
a
b
c
```

Decorator

OLP:983

- a callable that “mangles” an object (function,class, whatever)
  - usually a wrapper around an original object

goo.gl/FOIBBn

```
import datetime
def logthis(f):
    def wrapper(*args,**kwargs):
        start=datetime.datetime.now()
        ret=f(*args,**kwargs)
        end=datetime.datetime.now()
        print "Runtime:",end-start
        return ret
    wrapper.__name__=f.__name__
    return wrapper
```

```
>>> @logthis   ## like foo=logthis(foo)
... def foo(x):print x
>>> foo('Hello')
Hello
Runtime: 0:00:00.000043
```

```
class decowithargs(object):
    def __init__(self,before,after):
        self.before=before
        self.after=after
    def __call__(self,f):
        def wrapper(*args,**kvargs):
            return self.before+
              f(*args,**kvargs)+
              self.after
        return wrapper
```

```
>>> @decowithargs('BB','EE')
def foo2(s):return s.upper()
>>> foo2('apple')
BBAPPLEEE
```

Performance tips

- Membership testing
  - Use set or dict instead of list
    - b.has_key[a] instead of a in b
- Loops
  - use iterators (dict.iter*)
  - use generator, nested generators
  - prefer comprehensions

```
# Local variables accessed faster
# Avoid dots, or use map instead
upper = str.upper
newlist = []
append = newlist.append
for word in oldlist:
   append(upper(word))
```

```
>>> import profile
>>> profile.run('f()')
         400000 function calls in 2.786 seconds
   Ordered by: standard name
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    99999    0.358    0.000    0.358    0.000 :0(cos)
    99999    0.334    0.000    0.334    0.000 :0(random)
        1    0.000    0.000    0.000    0.000 :0(setprofile)
    99999    0.357    0.000    0.357    0.000 :0(sin)
    99999    0.345    0.000    0.345    0.000 :0(tan)
```

goo.gl/xmwnY

Python OOP

- old and new type of classes
  - type(name,bases,dict) is the glue (like bless in perl)
  - python 3.x has “new-type” only

OLP:625

```
# Old style class
>>> class A:
...      def f():pass
>>> A.i=1
>>> A.__dict__
{'i': 1,
'__module__': '__main__',
'__doc__': None,
'f': <function f at 0xb76ab374>}
>>> a=A()
>>> a.i
1
>>> a.__dict__
{}
```

```
>>> a.i=2
>>> a.j=3
>>> a.f=4
>>> a.__dict__
{'i': 2, 'j': 3, 'f': 4}
>>> a.f()
TypeError:
 'int' object is not
callable
>>> type(A)
<type 'classobj'>
>>> type(a)
<type 'instance'>
```

```
# New style class
>>> class B(object):
...      __slots__=('i')
...      def f():pass
>>> B.j=1
>>> b=B()
>>> b.k=1
AttributeError:
'B' object has no attribute 'k'
>>> b.j=1
AttributeError:
'B' object attribute 'j' is read-only
>>> b.f=1
AttributeError:
'B' object attribute 'f' is read-only
>>> type(B)
<type 'type'>
>>> type(b)
<class '__main__.B'>
```

OOP - Member functions

- constructors (__new__, __init__), destructor (__del__), weakref
  - use __new__ if extending a inmutable
- member functions (the magic 'self')

```
>>> class mystr(str):
      def __new__(klass,value):
        self=str.__new__(klass,
             str.upper(value))
        return self
>>> mystr('apple')
'APPLE'
```

```
>>> class C(object):
    def __new__(*args): print "New:",args
    def __del__(*args): print "Del:",args
>>> c=C(1,2)
New: (<class '__main__.C'>, 1, 2)
>>> c=1
## Nothing is printed
>>> class D(object):
    def __init__(*args): print "Init:",args
    def __del__(*args): print "Del:",args
>>> d=D(3,4)
Init: (<__main__.D object at 0x9a1b48c>,3,4)
>>> d=1
Del: (<__main__.D object at 0x9907f4c>,)
```

```
>>> class C(object):
  def __del__(self): print "Die"
>>> c=C()
>>> c
<__main__.C at 0x9a913ec>
>>> wr=weakref.ref(c)
>>> wr()
<__main__.C at 0x9a913ec>
>>> del(c)
Die
>>> wr
<weakref at 0xb744cfcc; dead>
```

OOP - Properties

- properties
  - static properties
  - internal dict
  - dynamically added (except __slots__)
  - OLP:792 only runtime check for private/public
    - A.__var : not available outside (but A._A__var works !)
  - use of helper classes
    - OLP:948 built in property class (getter,setter,deleter)
    - self made (__get__,__set__,__del__)

```
class A(object):
  def __init__(self):
      self.__class__.count+=1
```

```
class Human(object):
   __slots__=('__name')
   def __init__(self,name):
     self.__name=name
   def __getName(self):
     return self.__name
   def __setName(self,value):
     raise AttributeError("Cannot change name")
   name=property(__getName,__setName)
```

```
>>> joe=Human('Joe')
>>> joe.name
'Joe'
>>> joe.name='Jack'
AttributeError: Cannot change name
```

Python OOP - operators

- operator overload
  - __add__, __radd__
  - __str__, __repr__
  - ordering: __lt__,__eq__ or __cmp__
  - iteration,Sequence : __len__, __next__
  - callable: __call__

```
class Person(object):
    def __init__(self,name,age):
        self.name=name
        self.age=age

    def __str__(self):
      return "%s/%d" %(self.name,self.age)
    def __cmp__(self,other):
        return cmp(self.name,other.name)
    def __add__(self,num):
        self.age+=num
        return self
    __repr__=__str__
```

```
>>> p=Person("John",9)
>>> p=p+11
>>> p2=Person("April",22)
>>> print sorted((p,p2))
[April/22, John/20]
```

Python OOP - decorators

OLP:705

- helper decorators
  - @classmethod
  - @staticmethod
  - abc -> @abstractmethod
    @abstractproperty

```
from abc import ABCMeta,abstractmethod
class Helper(object):
    __metaclass__=ABCMeta
    @abstractmethod
    def help(self): pass

class A(Helper):
    def help():pass

class B(Helper): pass
```

```
>>> a=A()
>>> b=B()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class B with abstract methods help
```

goo.gl/Y7ndx

Python class templates

OLP:1015

- metaclass
- abstract classes
  - collections.ABC (Sequence, etc)
- interface
  - just ducktyping

```
class MyMeta(type):
    def __new__(meta, classname, supers, classdict):
        classdict['__str__']=MyMeta.toString
     return type.__new__(meta, classname, supers, classdict)
    @staticmethod
    def toString(instance):
        out=[str(instance.__class__)]
        out.extend("%s/%s" % (var,value)
            for var,value in instance.__dict__.iteritems())
        return ",".join(out)
class Person(object):
    __metaclass__=MyMeta
```

```
>>> a=Person()
>>> a.name='john'
>>> a.age='99'
>>> print a
<class '__main__.Person'>,age/99,name/john
```

Lab 4: Python OO