COMS 3101-3 Programming Languages – Python: Lecture 5

Kangkook Jee

jikk@cs.columbia.edu

Agenda

- Standard Library
  - os, urllib2, pickle
- Database
- Regular Expression
- Debugging
  - PDB, tracing
- Unit testing

OS

python interface to OS operations

File Operation with ‘os’

- ‘os’ module defines interfaces that enable interactions with operating systems
  - Most frequently used component of standard library
  - Implements majority subset of OS system call API

```
>>> import os
>>> os.system('date')  # OS specific command
Wed Sep 9 22:16:59 EDT 2013
0
```

- ‘os.path’ sub-module defines interfaces for filename manipulation

```
>>> os.path.isdir("/tmp")  # some folder
True
```

os.path Module – manipulate pathnames

- `os.path.abspath(path)`: Returns the absolute pathname for a relative path

```
>>> os.path.abspath('python')
'/opt/local/bin/python'
```

- `os.path.basename(path)`: Returns the absolute pathname for a relative path

```
>>> os.path.basepath('/opt/local/bin/python’)
‘python'
```

- `os.path.getsize(path)`: Returns the size of path in byte

```
>>> os.path.getsize("python")
13404
```

- `os.path.isfile(path)`: Returns True if the path points to a file
- `os.path.isdir(path)`: Returns True if the path points to a directory

os Module – list, walk content of a directory

- `os.listdir(path)` lists files in a directory

```
>>> os.listdir("/tmp")
['.font-unix', '.ICE-unix', ... , android-jikk']
```

- `os.walk(path)` returns generator object traverse sub-directories in depth-first fashion

```
>>> w = os.walk('/tmp')
>>> loc = w.next()
>>> while w:
...     print loc
...     loc = w.next()
```

collections

High-Performance Container Data -types

collections.defaultdict

- A dictionary class that automatically supplies default values for missing keys
- Is initialized with a factory object, that create
  - can be a function or a class object
  - can be a basic type (`list, set, dict, int` initializes to default value)

Counter using `dict`

```
 1 def count_seq(seq):
 2     seq_dict = {}
 3     for ent in seq:
 4         if ent in seq_dict:
 5             seq_dict[ent] += 1
 6         else:
 7             seq_dict[ent] = 1
 8     return seq_dict

>>> count_chr('sdfs')
{'s': 2, 'd': 1, 'f': 1}
```

Counter using `defaultdict`

```
10 from collections import defaultdict
11
12 def count_seq0(seq):
13   seq_dict = defaultdict(int)
14   for ent in seq:
15     seq_dict[ent] += 1
16   return seq_dict

>>> count_chr('sdfs')
defaultdict(<type 'int'>, {'s': 2,
'd': 1, 'f': 1})
```

collections.Counter

- Easy interface to count hashable(immutable) objects in collections (often strings)
- Once created, they are dictionaries mapping each object to its count
- Support method `most_common(n)`
- Can be updated with other counters or dictionaries

```
>>> from collections import Counter
>>> c = Counter('banana')
>>> c
Counter({'a': 3, 'n': 2, 'b': 1})
>>> c.most_common(2)
[('a', 3), ('n', 2)]
>>> c.update({'b':1})
>>> c
Counter({'a': 3, 'b': 2, 'n': 2})
>>> c['b']
2
```

urllib2

Open / Access resource by URL

urllib2: Fetch URLs

- URL: uniform resource locator
  - Support various Internet protocols: http, ftp, file …
- urllib2 enables
  - fetch internet resources located by URL
  - Interface to modify request headers

```
#URL for file
file://localhost/Users/jikk/jikk_web/index.html

#URL for HTTP
http://www.cs.columbia.edu/~jikk/index.html

#URL for ftp
ftp://user:password@host:port/path
```

urllib2: Getting Contents

- `urllib2.Request()`
  - returns URL Request object that you are making
  - Can specify extra information(metadata) associated to the request
    ex) request type, browser type, cookie
- `urllib2.urlopen()`
  - Open connection to the host and return response object

```
#basic usage
import urllib2
req = urllib2.Request("http://www.columbia.edu/index.html")
url = urllib2.urlopen(req)
info = url.info()
#getting meta-data (file size)
info.getheaders("Content-length")
['5068']

#read file
lines = url.readlines()
```

urllib2: Download files

- URL response operates as file object
  - Can read and write its contents to another file object
- `shutil` module provides easier interfaces to manipulate files

```
#downloading large binary file
url = urllib2.urlopen("ftp://ftp.sec.gov/edgar/xbrldata.zip")
output = open("output.zip", "wb")

CHUNK_SIZE=1024
buf = url.read(CHUNK_SIZE)
len(buf)
while len(buf):
    output.write(buf)
    buf = req.read(CHUNK_SIZE)
    output.write(buf)

#copying file using shell utilities(shutil)
import shutil
url= urllib2.urlopen("ftp://ftp.sec.gov/edgar/xbrldata.zip")
output = open("output1.zip", "wb")
shutil.copyfileobj(url, output)

output.close()
```

pickle

Object serialization / Data persistence

Pickle: Object Serialization

- Provide a convenient way to store Python objects in file and reload them
- Allows saving/reloading program data or transferring them over a network
- Can pickle almost everything
  - All standard data types
  - User defined functions, classes and instances
  - Works on complete object hierarchies
  - Classes need to be defined when un-pickling

```
import pickle
f = open('zip2addr.pickle','w')
pickle.dump(zip2addr, f)
f.close()
```

```
f = open('zip2addr.pickle','r')
zip2addr = pickle.load(f)
f.close()
```

Pickle: Protocols and cPickle

- Normally pickle uses a plaintext ASCII protocol
- Newer protocols available
  - 0: ASCII protocol
  - 1: old binary format (backward compatible)
  - 2: new protocol (≧ Python 2.3, more efficient)
- `cPickle`: More efficient re-implementation of Pickle in native C
  - Always use this for large object hierarchies (up to 1000x faster)

```
import cPickle as pickle
f = open('zip2addr.pickle','wb')
zip2addr = pickle.dump(zip2addr, f, protocol=2)
f.close()
```

DATABASE

Relational Database Management Systems

- Software to manage a large collection of data in digital form
- Takes care of accessing/storing data efficiency (indexing)
- Make accessing data easy
- Data representation is relational (tables)

Knight table:

id	name	title	favorite_color
0	Bedvere	the wise	Blue
1	Lancelot	the brave	Blue
2	Galahad	the pure	Yellow

Quest table:

id	quiest
0	Bedvere

- Use a special query language (most common: SQL)
- Example RDBMS: MySQL, Oracle, SQLite ...

Setting up a simple SQLite database

- Tiny, efficient database manager
- Delivered with Python (mac, linux)
- Python wrapper supported

```
jikk$ sqlite3 test.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
sqlite> .quit
```

Structured Query Language(SQL): Creating tables

- Most DBMS can be queried / manipulated using SQL
- SQL is another language to learn
  - Tons of tutorials / documentation online
- Can just type SQL line into the Sqlite prompt
- Example: Creating table structure

```
CREATE TABLE fmarket (
    fmid INT PRIMARY KEY,
    name VARCHAR(1000),
    x    FLOAT,
    y    FLOAT
);
```

FMID	NAME	X	Y

CREATE TABLE command defines table schema (structure)

ALTER TABLE , DROP TABLE command supported

Structured Query Language(SQL) – Populating tables

```
INSERT INTO fmarket VALUES (
    1234, 'Market A', -76.13, 36.84
);

INSERT INTO fmarket VALUES (
    1235, 'Market B', -93.25, 45.00
);


INSERT INTO fmarket VALUES (
    1236, 'Market C', -98.48, 45.46
);
```

FMID	NAME	X	Y
1234	Market A	-76.13	36.84
1235	Market B	-93.25	45.00
1236	Market C	-98.48	45.46

UPDATE FROM ... : updates table entries

DELETE from ... : deletes table entries

Structured Query Language(SQL) – Selecting Data

- SELECT ... FROM statements to select/display data

FMID	NAME	X	Y
1234	Market A	-76.13	36.84
1235	Market B	-93.25	45.00
1236	Market C	-98.48	45.46

```
sqlite> SELECT * FROM fmarket
   ...>      WHERE x <= -80 and y > 40;

1235         Market B    -93.25       45.0
1236         Market C    -98.48       45.46
```

Opening and using the database in Python

```
>>> import sqlite3
>>> conn = sqlite3.connect("farmers.db")
>>> cursor = conn.cursor()
>>> cursor.execute("SELECT * FROM fmarket
        WHERE x <= -80 and y > 40")
<sqlite3.Cursor object at 0x1092ed030>
>>> cur.fetchone()
(1235, u'Market B', -93.25, 45.0)
>>> cursor.execute("SELECT * FROM fmarket
        WHERE x <= -80 and y > 40”)
>>> for entry in cursor:
...     print entry
...
(1235, u'Market B', -93.25, 45.0)
(1236, u'Market C', -98.48, 45.46)
```

Database vs. File system

sqlite3

- Structured data with query language
- Fast lookup due to indexing
- Need to transform program data structure to RDBMS schema
- Support advanced operations
  - ex) Joining tables
- Data consistency and security

pickle

- Non-structured data access
- Linear search
- Preserved data structures defined from program
- No DBMS overhead
- Sometimes, works better for small dataset

REGULAR EXPRESSION

Some materials are courtesy of Google python tutorial

Regular Expressions

- Need to match and search complex patterns in text
- Regular Expressions (RegEx) are used to describe sets of strings
- Formally, can be used to define any regular language
  - Languages recognized by finite state automata left/right recursive grammar

```
>>> import re
# RegEx to match Columbia UNIs
... uni_re = re.compile("[A-z]{2,3}[0-9]{4}")
>>> uni_re.search("My UNI is: xyz1234")
<_sre.SRE_Match object at 0x106da9e68 >.
>>> uni_re.search("My UNI is xy")
>>> uni_re.search("My UNI is 1234xy")
>>> uni_re.search("My UNI is xyab1234")
```

Regular Expressions: Patterns

- Any literal character is a regular expression
  Regular characters (a, X, 9 …): just matches themselves
- Meta characters defined for extended pattern matching
  Complete list: `. ^ $ + ? { } [ ] \ | ( )`

.	matches any single character but '\n'
AB	regular characters matches themselves
|	Or operator A | B | C : 'A' or 'B' or 'C'
[ ]	set of characters [ABC] : 'A' or 'B' or 'C' , [a-Z]: alphabets, [a-z0-9]: lowercase alphabets + digits
^ $	beginning / end of a line
\	escape character

- Any meta characters need to be escaped with \ when use literally
  pattern `'\[abc\]'` matches string `'[abc]'`
- You can use meta characters Square brackets ([,]) without '\'
  pattern [.a\] matches '.' or 'a' or '\'

Regular Expressions: Pattern examples

Special sequences

`\d`	any decimal digits	`[0-9]`
`\D`	any non-decimal digits	`[^0-9]`
`\s`	any whitespace characters	`[ \t\n\r\f\v]`
`\S`	any non-whitespace characters	`[^ \t\n\r\f\v]`
`\w`	any alphanumeric characters	`[a-zA-Z0-9_]`
`\W`	any non-alphanumeric characters	`[^a-zA-Z0-9_]`

- Some of special sequences beginning with '\' represent predefined sets
- ^ operator (inside [ ]) negates the following pattern
  - everything but …

Regular Expressions: Operators

- If R and S are regular expressions
  - RS is a regular expression: R followed by S
  - R|S is regular expression: either R or S
- If R is regular expression
  - (R) is a regular expression: grouping
  - R? is a regular expression: 0 or 1 repetition of R
  - R+ is a regular expression: 1 or more repetition of R
  - R+ is a regular expression: 0 or more repetition of R
  - R{n} is a regular expression: n repetition of R
  - R{n, m} is a regular expression: at least n and at most m repetition of R

Example

```
a(b|c)* matches ‘a’ followed by arbitrary sequence of ‘b’
and ‘c’

a, ab, ac, abbb, abcb ...
```

Regular Expression in Python: re module

- Pattern has to be raw string – string prefixed with 'r'
  ex) r"[abc]", r"\d\s*\d"
- Apply pattern to different match functions
  - `re.match(pat, string)`: check if pattern matches for the entire string
  - `re.search(pat, string)`: searches for an occurrence of pattern
  - both function returns *`match` object* if match is found, otherwise None
- Substitution
  - `re.sub(pat, repl, string)`: returns a string obtained by replacing pattern with `repl`

```
>>> re.match(r"ab*", "abbbcd")
<_sre.SRE_Match object at 0x106da9e68 >
>>> re.match(r"ab*", "cdabbbc")
>>> re.search(r"ab*", "cdabbbc")
<_sre.SRE_Match object at 0x10df244a8>
>>> re.sub(r"ab*", "--", "cdabbbc")
'cd--c'
```

Regular Expression in Python: The re module

- Can pre-compile the regular expression into a pattern object(matcher) for efficiency
  - `re.compile()`
- `finditer(re, string)` returns an iterator over all on-overlapping matches

```
>>> matcher = re.compile(r"ab*")
# equivalent to re.finditer(r"ab*", "cdabbabc")
>>> list(matcher.finditer("cdabbabc"))
[<_sre.SRE_Match object at 0x1096dbd98 >, <_sre.
    SRE_Match object at 0x1096dbe00 >]
```

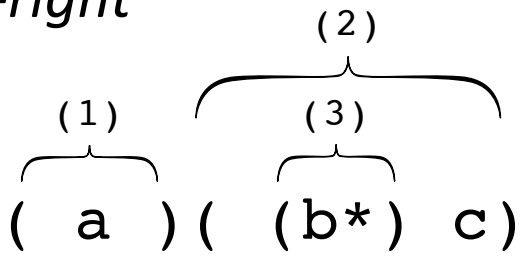
Regular Expression in Python: The re module

- Match object contains:
  - Positional information about the match in the string
  - match.start(), match.end()

```
>>> matcher = re.compile(r"ab*")
# equivalent to re.finditer(r"ab*", "cdabbabc")
>>> match_list =  list(matcher.finditer("cdabbabc"))
>>> for m in match_list:
...     print m.start(), m.end()
...
2 5  # cdabbabc
5 7  # cdabbabc
```

Regular Expression in Python: The re module

- Match object contains copies of subsequences of the string
  - Corresponding to ( ) groups in regular expression
  - Groups are indexed *outside-in, left-to-right*




```
>>> match = re.search(”(a)((b*)c)","cabbbc")
>>> match.group(0) # Entire match
'abbbc'
>>>  match.group(1)
'a'
>>>  match.group(2)
'bbbc'
>>> match.group(3)
'bbb'
```

Regular Expression: Search Ambiguity

- What if `re.search()` finds multiple matches?

```
## i+ = one or more i's, as many as possible.

match = re.search(r'pi+', 'piiig') =>  found, match.group() == "piii"

## Finds the first/leftmost solution, and within it drives the +
## as far as possible (aka 'leftmost and largest').

match = re.search(r'i+', 'piigiiii') =>  found, match.group() == "ii"
```

- Leftmost and Largest rule
  1. First, the search find the leftmost match for the pattern
  2. Second, use up as much of the string as possible (greedy)

DEBUGGING AND TESTING

Programs are Error Prone

- Syntax errors ←Detected by interpreter
- Errors at runtime ← Exception handling
- Incorrect programming behavior(wrong result)
  - Sometimes works, sometime not

Debugging and Testing

Debugging Tips for Scalable Project

- Utilize Python Interpreter actively
  - Program should be organized into functions / classes of reasonable sizes
- Log / Trace program behavior
  - Using print() or logging module
- assert() everywhere
- Be familiar with pdb (Python debugger)
- Establish unit testing framework

Debugging with `print`

- Most common way of debugging: Simply print intermediate results
- Prints all relevant information and reference to which part of the program prints
- Better: Write debugging statements to `sys.stderr`
- Comment / uncomment debugging code

```
def mirror(lst):
    ret = []
    for i in range(len(lst)):
        ret.append(lst[-i])
        #print >> sys.stderr, \
        # "mirror: list for i={0}: ".format(i)
        #print >> sys.stderr,"{1}\n".format(lst)
    return lst + ret

x = [1,2,3]
print(mirror(x)) # Expected: [1,2,3,3,2,1]
```

logging module for debugging

- logging module to log errors and debugging messages
- Provides central control over debugging output

```
import logging
logging.basicConfig(level = logging.DEBUG)

def mirror(lst):
  ret = []
  for i in range(len(lst)):
    ret.append(lst[-i - 1])
    logging.debug("list for i={0}: {1} ".format(i, lst[-i - 1]))
  return lst + ret
```

```
>>> mirror([1,2,3])
DEBUG:root:list for i=0: 3
DEBUG:root:list for i=1: 2
DEBUG:root:list for i=2: 1
[1, 2, 3, 3, 2, 1]
```

logging – Logging Levels

- Can output messages to on different logging levels
  - Output messages of *LEVEL* and above

```
logging.basicConfig(level=logging.LEVEL)
```

level	function
`logging.CRITICAL`	`logging.critical()`
`logging.ERROR`	`logging.error()`
`logging.WARNING`	`logging.warning()`
`logging.INFO`	`logging.info()`
`logging.DEBUG`	`logging.debug()`

logging – more on logging

- Can output messages to a log file

```
logging.basicConfig(level=logging.DEBUG,
                    filename = 'bugs.log')
```

- Config is valid for all modules in a program
  - Only set logfile and level once in main
- Can add and time

```
logging.basicConfig(level=logging.DEBUG,
                        filename='bugs.log',
                        format='%(acstime)s %(message)')
```

- More on logging
  http://docs.python.org/library/logging.html

Python debugger - pdb

- Python provides a built-in debugger (module pdb)
- Allows to execute code line-by-line
- pdb allows access to program state

- Postmortem debugging

```
$ python –m pdb mirror.py
```

- Or launching pdb interactively from Python console

```
>>> from mirror import mirror
>>> import pdb
>>> mirror([1, 2, 3])
>>> pdb.pm()
```

pdb Commands

- b: set breakpoint
- n: next line
- r: return from the function
- l: source code for current file
- c: continue execution until next breakpoint

Using assert

- Allow you to insert simple tests in the code

```
assert condition [, expression]
```

- Interpreter evaluates condition
  - if condition is True, nothing happends
  - if condition is False, an AssertionError is raised with expression as argument
- Can use assertion to document logical invariants

```
def eat(self, item):
    assert isinstance(item, Food)
    self.stomach.append(item)
```

Unit Testing

- Need to test various units in a module(each function, method or class ...) independently
- Tests can fail for various reasons
- More ‘permanent’ solution to check if code is still working as desired if parts are changed
- Goes hand-in-hand with design specification
- unittest module comes with Python

Defining Test Cases

- A test case is an elementary unit of tests (tests a single class, method, or function)
- *Methods*: individual tests
- Special method `setUp(self)` is run before every test to set up the test environment
- Special method `tearDown(self)` performs cleanup after a test
- Test cases are classes with base class `unittest.TestCase`

```
class TestMyInt(unittest.TestCase):
    def setUp(self):
        print("Prepare testing")

    def tearDown(self):
        print("Clean-up testing")

    def testArith(self):
        self.assertEqual(MyInt(1) + MyInt(2), MyInt(3))
        ...
```

Defining Test Cases – assert Methods

- instances of classes with base `TestCase` have `assert*` methods, which will perform tests

Method	meaning
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNone(x)`	`x is None`
`assertIn(a, b)`	`a in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`

Running Tests

- Can invoke test runner from within the module containing testcases
  - Test all defined cases

```
if __name__ == '__main__':
    unittest.main()
```

- Can also run tests from the command line

```
$python –m unittest tested_module
```

- Can run a specific test case

```
$python –m unittest tested_module.TestCase
```

backup slide

Final Projects & Demos

- Due on Oct 17th
  - Submission via Courseworks
- Submit a single compressed archive containing
  - Source code and Epydoc output: Will learn about it next week
  - Project summary: A write-up describing your project, results, lessons learned.
  - README: Explains about how to run your program and its functionalities
- Demos
  - Schedule a short demo for 10 ~ 15 minute via Doodle