Lecture 1 – 4/2/13

[boverly@pce.uwedu](mailto:boverly@pce.uwedu)

Need a project proposal by next week.

<https://github.com/PythonCHB> - main python course code

ssh [hhoaglundbiron@teampython.west.isilon.com](mailto:hhoaglundbiron@teampython.west.isilon.com) , password = a

Unit Testing

<http://docs.python.org/2/library/unittest.html> or Nose

A string under the name of a function can be accessed with <function\_name>.\_\_doc\_\_

Unit tests can be added to a test class (no matter the name), as long as the test/function has “test” in the name.

Lecture 2 – 4/9/13

Project GUI library possibility: tk (python version of tcl)

Project Design: Could be just empty classes and methods, could be lines and bubbles

Make sure pdb is installed in Cygwin before next week

More Unit Testing

\_\_init\_\_.py – Read up on it! He glossed over it a bit…

Nose and pytest notes will be up on the website

Decorators

Applications:

* Profiling
* Memoization (caching)
* Frameworks
  + Testing
  + Web (pylons!)

Functions are first class objects!

* Python namespace is a dictionary – assigning a new function to a function called “foo” does not erase its value in the namespace, just assigns the key (“foo”) a different value. If something still references the original value, it **doesn’t get garbage-collected**.
* Functions can contain classes and other functions, or even return an inner function. When returning an inner function, calling that returned function actually calls the inner function, not the outer. In addition, this inner function is re-defined every time the outer function is run.
  + - A function (its code) and context is called a **closure**.

Decorator

* A decorator passes the defined function under it into the decorator function, so that when the defined function is called, it is actually calling the decorator function. It’s a wrapper!
* Multiple decorators can be used at once, but **order matters**.

Lecture 3 – 4/16/13

Lambda functions

Example:

def fact\_with\_lambda(n):

return reduce(lambda x, y: x \* y, range(2, n+1), 1)

Same as:

def fact\_no\_lambda(n):

return reduce(mul, range(2, n+1), 1)

Python debugging and PDB

\_\_repr\_\_

Can be used if you want a class to be represented as a particular string in debugging.

sys module

sys.exc\_info(): information about current excepton in processor

sys.\_getframe(depth\_int): get stack frame at specified depth

sys.settrace(my\_trace\_func): set tracing function, can have certain things happening when certain events fire, specifically:

* + - enter (call)
    - leave (return)
    - next line
    - exception
    - etc.

PDB

Run with (one option of many):

python –m pdb *thing\_to\_be\_debugged*

Interesting commands:

**list**: gives the source code

Any python code works here!

Lecture 4 – 4/30/13

SQL

Examples, etc @ (git clone) http://github.com/cewing/training.python\_sql.git

One-to-One relationships

* They may be in separate tables depending on how often you want to access each table, or user-permission related

Many-to-Many relationships

* May include a “join” table to indicate which keys go with which other keys in separate tables

DB-API

* Globals: apilevel, threadsafety, paramstyle – particularly look at threadsafety! **(sqlite3 is 1, look into this!!)**
* Don’t use the output from .execute()! Undefined.
* Results from a cursor can be used with .fetchone(), .fetchall(), etc.
* **NUMBER** may be better to use than INTEGER, and **STRING** rather than whatever else…

Python “with” statements may be useful for me!

* It both opens, closes AND commits (and rollback) when the block is closed.

Sqlite3

* You have to turn foreign keys on… PRAGMA foreign\_keys = ON;
* Any operations in Data Manipulation will need an explicit .commit() call.

Python “raise” – raise “e” will show THAT line instead of the original location where it was raised.

Isolation levels **(potentially interesting for multiprocessing)**

* DEFERRED - default
* IMMEDIATE
* EXCLUSIVE
* Can be set with isolation\_level=”BLAH” in connect call

**QUESTION**: Can these ready event type things happen in multiprocessing?? Does threaded.py work with multiprocessing?

**Python Module of the Week – pymotw!**

Lecture 5 – 5/7/13

Review

ipython, ipdb

Advanced OO

* New style classes need to inherit from object (class C(object):)
* Adding “?” at the end of things (say, in the interpreter), will give all the docstrings

Lecture 6 – 5/14/13

C Extensions in Python

* You can export method(s) from Python using distutils, etc
* It’s possible to have “virtual environments” within a folder in python with virtualenv, where only some packages will be installed within it, and it won’t interfere with your system setup.
* Virtualenv
  + Commands:
    - virtualenv /path/to/env/folder
    - Source /path/to/env/folder/bin/activate
    - Deactivate
* All we care about is the .so files at the end.

Lecture 7 – 5/21/13

Threading and Multiprocessing

* Went over argparse
* Multithreading is good for IO things – multiprocessing is necessary otherwise. The GIL gets in the way.

Lecture 8 – 5/28/13

Performance and Profiling

* timeit: Comparing two python statements’ execution time
* time.clock() on Windows rather than time.time()!
* Profiling: “cProfile” has low overhead, same API as “profile”
  + python -m cProfile code.py
  + save profiles with “-o dump.profile” etc
  + will give an overview of how long each function in your code takes to run
  + pstats can parse your profile for more readability
  + Run Snake Run – gives you a visual of your profile (treemap)
* Note slide about boosting Python performance
  + += is slower than ‘’.join([<list of strings>])
  + map/list comprehensions faster than loops