Python II

Introductions

- Name
- Role/Job
- Length of tenure
- Programming experience
  - Python specifically
- Why did you sign up?
- Define success for you in this course

What this class is

- Intermediate / Advanced topics in Python
- Lots of hands-on lab work
- Lots of class participation
  - Share your war stories and experiences
- A walk through some real-world usages for the language
- The Alpha release, and delivery, of the course material

What this class isn't

- In depth information on specific Python modules
- Every intermediate/advanced topic in Python
- Rigidly structured
- Finished yet...

Alpha - My ask of you

- Feedback!
- What's working?
- What's not working?
- What would you change?
- What knowledge was too deep?
  - Too shallow?
- What topics were good, bad, ugly?

Class Structure

- We have a lot of topics we can cover in this class
- I want to focus on the topics that are important to you
- Each of the topics is (mostly) standalone
  - We will work "modules" in the course in the order voted on by you (TBD)

Socrative

- We will be using Socrative to poll/quiz/register discontent
- http://socrative.com/
  - Choose "Student Login"
  - Enter my Room code: JEREMYPRICE

Pre-Work

System Config

- Make sure Python is installed
  - Python 2 for this course
- Lab machines have CentOS 7 with Python 2 already installed

```
python --version
```

Editor Config

- Install and setup your favorite editor
  - vim, atom.io, emacs, nano, etc.

Virtualenv

- virtualenv allows us to setup isolated python environment customized to your version and dependency requirements
- Comes with distribute, easy_install, and pip
- Allows control of environments where root authority is lacking

```
#install on CentOS 7
sudo yum install -y python-virtualenv

# create a new virtual env
virtualenv <directory_name>

# work in the new virtual env
cd <directory_name>
source bin/activate

# stop working on the virtual env
deactivate
```

PIP

- PIP is a Python package manager
  - built on setuptools package
  - finds packages in the cheese shop
    - https://pypi.python.org/pypi
- Standard package commands:
  - install, uninstall, list, search
- Installation instructions:
  - Already installed by virtualenv!

IPython

- We will use IPython in this class for a lot of our shell interaction
- You can create a virtualenv if desired
  - IPython, and dependencies, won't be the only packages we will install

```
# make sure you are in the virtualenv
pip install ipython
```

Lab

- Verify Python version info
- Install virtualenv
- Create a virtualenv
  - activate it
- Install IPython in tthe virtualenv
- Register your completion in Socrative

Python Review

Anatomy of a Module

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
# Line #1 = shebang
# Line #2 = encoding declaration

"""Docstrings"""
# Inline documentation

Import(s)

Statements
```

Python Keywords

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Tuple Type

- Tuples are sequences of other objects that cannot be changed (they are “immutable”)
- Tuples can contain any type of object, and can be sliced (remember, they’re ordered)
- A one element tuple is formed by (element,) where the comma makes it a tuple
- Examples:

```
tup = (1, "a", 3)
tup = tuple()
tup = (1,) # Comma needed for single element
gen = (x**2 for x in range(1,4)) # Not a tuple!
```

List Type

- Lists are sequences that can be changed (“mutable”)
- Lists can contain other object types and be sliced
- A list is a set of objects separated by commas and closed in square brackets
- Examples:

```
lst = [1, "a", 3]
lst = list()
lst = [x for x in xrange(5)]
```

Dictionary Type

- A dictionary (“dict”) type is a set of expressions that are associated 1 to 1 with other expressions
- A dict provides a handy “mapping” between a “key” expression and its associated “value” expression
- A dict is not an ordered sequence, it’s a mapping.
- Examples:

```
dic = {"a": 1}
dic = dict(a=1)
dic = {x: ord(x) for x in range(97, 107)}
```

Set Type

- A set is a mutable group of unordered immutable objects with unique values i.e. no duplicates (so the set is mutable, but not the objects within it)
- Uses the '{}' symbols just like a dictionary, but doesn't have ':' i.e. no key/value pairs
- Slicing is not allowed (why?), but iterators work
- Set operations are powerful, but perhaps not the most efficient
- Examples:

```
st = {1,2,2,3}
st = set(1,2,2,3)
st = {x for x in range(1,4)}
```

Yield Statement

- yield is similar to return, but suspends execution of the called function instead of ending the function
  - Saves the state
- On the next call to the function, yield picks up where it left off, with all identifier values still holding the same values
- A function with a yield statement is a generator
- Example:

```
def foo(bar=10):
    i = 1
    while i < bar:
        yield i
```

With Statement

- The with statement is used to run a suite of other statements under a “context manager”
- Special methods __entry__() and __exit() are called to setup and takedown a “context”
- Common for doing i/o (which auto-closes the file handle) and threading (which auto-acquires and releases lock)

```
with file("foo.txt", "r") as infile:
    all_txt = infile.read()
```

Import and From Statements

- To make a set of identifiers in another module available for use by your module, you must first use import or from
- import pulls in identifiers from module(s) but the module name must be used as a prefix
  - only module name is added to local namespace
- from pulls in identifiers from modules but avoids need to prefix with the module name
  - identifier is added to local namespace

Lab - Pyreview

- Follow the instructions in:

`lab_pyreview.py`

- Register your completion in Socrative

Serialization

JSON

- Javascript Object Notation
- Part of Javascript language definition
- Easy for parse for humans and machines
- Language independent
- “lightweight data interchange format”

```
Name/value pair collection (dict in Python)
Ordered list of values (list in Python)
```

JSON Details

- An “object” is an unordered set of name:value pairs (called “members”) separated by commas and surrounded by curly braces
  - `{ Name1:Value1, Name2:Value1,… }`
- An “array” is a set of “elements” surrounded by square brackets
  - `[ element1, element2,… ]`
- Elements are strings, numbers, true, false, null, object, array

More JSON

- Strings are made of:
  - Unicode chars (\uhhhh)
  - \,\\,/,\b,\f,\n,\r,\t
- Numbers are signed integer, decimal, or exponent (“e”) flavors only
- Whitespace is fine

Python to JSON

- import the json module
- Serialize a Python object to JSON

```
import json
json.dump(python_obj, fo, **kwargs)
json.dumps(python_obj, **kwargs)
```

Note: JSON is not a "framed" protocol
i.e. can't append multiple JSON objects to same file

JSON to Python

- import the json module
- Deserialize a JSON object to Python

```
import json
python_obj = json.load(fo, **kwargs)
python_obj = json.loads(string, **kwargs)
```

JSON Module Caveats

- JSON keys in key/value pairs are always strings. Unlike Python.
- Default json module encoder only encodes “ASCII-compatible” strings. Use u for other encodings
- Same name in name:value pairs uses the last one
- Out of range floats are handled correctly (nan, inf, -inf)

Python to JSON Translations

```
dict -> object
list, tuple -> array
str, unicode -> string
int, float -> number
True, False -> true, false
None -> null
```

Pickle and Shelve

- The pickle module does a similar job to JSON, but is Python specific
  - Not good for machine data interchange
  - Allows multiple pickled objects to be dumped to the same file
    - must be loaded in same order on way back
- The shelve module essentially provided a persistent dict for pickled objects in a database
- Opinion: Only use these for interchange within your own app

Performance

- The builtin json module can be slow
- Other pypi packages have C extensions to speed them up
  - simplejson, yajl, python-cjson, UltraJSON
- Opinion:
  - Don't load/save huge json objects
  - profile your app to make sure it is the json encode/decode

Lab JSON

- Follow the instructions in:
  - `lab_json.py`
- Register your completion in Socrative

REST

REST Principles

- Representational State Transfer
- A simple, client/server web services API currently in favor
- Way of locating and manipulating “resources” (usually XML or JSON documents) on a network
- Commonly based HTTP protocol (GET, POST, PUT, DELETE)
- Stateless (all state on client or server)
- Simple, predictable resource pathing scheme based on URL

More RESTing

- ReSTful services generally map a CRUD interface (Create, Read, Update, Delete) by URL mappings that embed data
  - e.g. GET v2/{tenant_id}/servers
  - HTTP POST -> Create
  - GET -> Read
  - PUT -> Update
  - DELETE -> Delete
- HTTP GET calls to a ReSTful service should not change state i.e. read only

OpenStack REST API

- Openstack uses ReST to implement it's user-controlled cloud provisioning service
- Requires a set of "endpoint" URL's which have service request data appended e.g.
- Adding "v2.0/tokens" to the Identity service "endpoint" URL, inserting login credentials, and POST'ing will return a "token" that allows use of the v2.0 API
- e.g. Adding "v2/{tenant_id}/servers/ips for the Compute service will return server IP addresses

OpenStack REST API

- See the Openstack API Reference at http://api.openstack.org/ for complete API
- Workflow to use Openstack API:
  - Obtain tenant id and API key and authenticate (24 hour timeout)
  - Extract token id and appropriate endpoint URL for the desired service from the response
  - Send API request(s) to the appropriate service endpoint(s) including the X-Auth-Token HTTP header for each request
  - If a 401 HTTP response occurs, re-authenticate

Accessing Web Data: urllib2

- Client functions to access URL's
- `import urllib2`
- urllib2.urlopen(url[,data]) is a common call signature
  - url is the URL of the target resource
  - data is to be sent to the server
- Only HTTP uses data currently
- If data exists
  - it must be URL encoded
  - HTTP GET becomes a POST

urllib2

- A “file-like object” is returned from urlopen() which can be accessed with file semantics (read, readlines, etc.)
- `Raises:`
  `URLError (subclass of IOError)`
  `HTTPError (subclass of URLError)`
- Also takes urllib2.Request objects, useful for including HTTP headers in a dict (or use Request add_header method)
- Can handle redirections, HTTP error responses, cookies, proxies, HTTP authentication

Lab urllib2 REST

- Follow the instructions in:
  - `lab_urllib2_REST.py`
- Bonus points: print your repos from GitHub
  - https://api.github.com/users/<your_username>/repos
- Register your completion in Socrative