Accelerating Information Technology Innovation

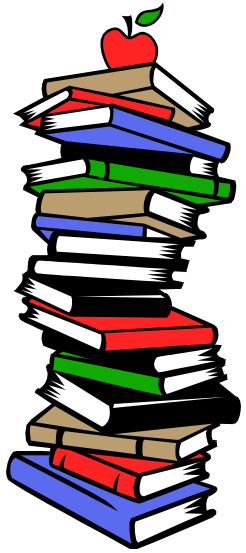
http://aiti.mit.edu

Ghana Summer 2011
Lecture 11 – Useful libraries and functions

MISTI

What else in Python?

- We've learned the rules and syntax of python.
  - Functions and variables
  - Classes
  - Exceptions
  - …
- What else?
  - How to re-use code
  - Useful Python libraries for common programming needs.



What if you wanted to…

- Search for a pattern in a string
  - E.g. Is there a substring `.oon` in `afternoon` where `.` represents any letter?
- Interact with the operating system
  - E.g. Make a sub-directory ‘project’ in your home directory
- Manipulating dates and times.
  - E.g. Determine the current date and format its output

- Implementing these functions would take significant time
- Python already has built-in code

Today

- How to re-use code
  - `import` and `reload` commands
- Search for a pattern in a string
  - `re` module
- Interact with the operating system
  - `os` and `shutil` modules
- Manipulating dates and times.
  - `datetime` and `time` modules

Modules and importing code

- Modules are python files that contain re-useable code
  - E.g. definitions of functions, classes, etc.
  - Create modules or use modules built in Python
  - Use a module with the `import` command

You've written a Python module athelete.py

athlete.py

```
class Athlete:
    def __init__(self, speed):
        self.speed = speed
    def get_speed(self):
        return self.speed
    def practice(self):
        speed += 1

john = Athlete(20)
mike = Athlete(15)
```

Re-use athlete.py

```
# Load athelete.py
>>> import athlete

# Access definitions in athelete.py
>>> athlete.Athlete
<class athlete.Athlete at …>
>>> athlete.john.getspeed()
20
>>> athlete.mike.getspeed()
15
```

Importing modules

- the import command loads the entire module

  `>>>` import athlete

  – leave out the `.py`

  – loads the Athlete class and instances mike and john

  – access definitions by including module name, i.e. `athlete.<name>`

- What about loading individual definitions?

  – E.g. load the Athlete class definition but leave out the instances mike and john

  – `from … import …`

Load only the Athlete class

```
# Load the Athlete class
>>> from athlete import Athlete

# Access the class directly instead of including the
   module name
>>> matthew = Athlete(30)
>>> matthew.getspeed()
30

# the john instance in athlete.py was not loaded
>>> john
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'john' is not defined
```

Today

- How to re-use code
  - `import` and `reload` commands
- Search for a pattern in a string
  - `re` module
- Interact with the operating system
  - `os` and `shutil` modules
- Manipulating dates and times.
  - `datetime` and `time` modules

Searching for a pattern in a string

- Use regular expressions
  - A convenient way of representing a pattern
  - Also known as REs, regexes, or regex patterns
- Support for regular expressions with the `re` module

Example REs

Normal words

1. `noon, dog, car` …

Wildcard characters

1. `.oon` represents any word that begin with any single character and ends with `oon`

Repeating characters

1. `abc?` represents `ab`, `abc`
2. `abc*` represents `ab`, `abc`, `abcc`, `abccc`, `abcccc`, …
3. `abc+` represents `abc`, `abcc`, `abccc`, `abcccc`, …

A class of characters

1. `[cfj]ar` represents `car`, `far`, or `jar`
2. `[a-m]ar represents aar, bar, car, dar, … mar`
3. `[^a-m]ar` represents any `.ar` word where `.` is not in `[a-m]`

Searching for REs

```
# Load re module
>>> import re

# Search for the first occurrence of a pattern in a string.
>>> m = re.search('[cjf]ar', 'Take care')
# Return a Match object
>>> m
<_sre.SRE_Match object at 0x...>

# Review the match
>>> m.group()       # The matched substring
'car'
>>> m.span()        # The indices of the match in the string
(5,8)

# None returned if there is no match
>>> m = re.search
>>> m
None
```

Matching vs. searching

Searching = look for an occurrence of a pattern anywhere in a string

>>> re.search(‘c’, ‘abcdef’)

<_sre.SRE_Match object at 0x…>

Matching = look for an occurrence that

>>> re.match(‘c’, ‘abcdef’)

<_sre.SRE_Match object at 0x…>

More REs…

```
# Search for an occurrence of the pattern at the beginn
>>> m = p.search(s)
# A Match object
>>> m
```

```
# Search for an occurrence of b

# Find all instances of the pattern
>>> p.findall(s)

# None if no match
>>> m = p.search('random string')
>>> m
None

# the re.match function searches
>>> re.match('[cjf]ar', s)

# Compile a pattern
>>> p = re.compile('[cjf]ar')
# A Pattern object
>>> p
<_sre.SRE_Pattern object at 0x...>
```

Today

- How to re-use code
  - `import` and `reload` commands
- Search for a pattern in a string
  - `re` module
- Interact with the operating system
  - `os` and `shutil` modules
- Manipulating dates and times.
  - `datetime` and `time` modules

os module

```
# Load the os module
>>> import os

# Return the current working directory
>>> os.getcwd()
/home/mike

# Change the current working directory to the Desktop sub-directory
>>> os.chdir('/home/mike/Desktop')
>>> os.getcwd()
/home/mike/Desktop

# Make a sub-directory 'project'
>>> os.mkdir('/home/mike/Desktop/project')
```

os.path submodule

Working with file paths

```
# Get the absolute path of a file or directory in the current
   directory
>>> os.chdir('/home/mike/Desktop')
>>> os.path.abspath('project')
/home/mike/Desktop/project

# Split an absolute file into a 2-tuple
# containing the parent directory and filename
>>> os.path.split('/home/mike/Desktop/project')
('/home/mike/Desktop', 'project')

# Join together a directory and a file name
>>> os.path.join('/home/mike/Desktop', 'project')
/home/mike/Desktop/project
```

shutil module

File copying and removal

```
# Copy roster file from mike’s directory to john’s directory
>>> shutil.copy(‘/home/mike/roster’, ‘/home/john/roster_copy’)

# Move roster file to the /tmp directory
>>> shutil.move(‘/home/mike/roster’, ‘/tmp’)

# Delete a directory all of its sub-directories
>>> shutil.rmtree(‘/home/mike/project’)
```

Today

- How to re-use code
  - `import` and `reload` commands
- Search for a pattern in a string
  - `re` module
- Interact with the operating system
  - `os` and `shutil` modules
- Manipulating dates and times.
  - `datetime` and `time` modules

datetime module