3. Modules

Modules are reusable libraries of code in Python. Python comes with many standard library modules.

A module is imported using the *import* statement.

```
>>> import time
>>> print time.asctime()
'Fri Mar 30 12:59:21 2012'
```

In this example, we've imported the *time* module and called the *asctime* function from that module, which returns current time as a string.

There is also another way to use the import statement.

```
>>> from time import asctime
>>> asctime()
'Fri Mar 30 13:01:37 2012'
```

Here were imported just the *asctime* function from the *time* module.

The *pydoc* command provides help on any module or a function.

```
$ pydoc time
Help on module time:

NAME
    time - This module provides various functions to manipulate time values.
...

$ pydoc time.asctime
Help on built-in function asctime in time:

time.asctime = asctime(...)
    asctime([tuple]) -> string
...
```

On Windows, the *pydoc* command is not available. The work-around is to use, the built-in *help* function.

```
>>> help('time')
Help on module time:

NAME
    time - This module provides various functions to manipulate time values.
...
```

Writing our own modules is very simple.

For example, create a file called *num.py* with the following content.

```
def square(x):
    return x * x

def cube(x):
    return x * x * x
```

Now open Python interterter:

```
>>> import num
>>> num.square(3)
9
>>> num.cube(3)
27
```

Thats all we've written a python library.

Try *pydoc num* (*pydoc.bat numbers* on Windows) to see documentation for this numbers modules. It won't have any documentation as we haven't providied anything yet.

In Python, it is possible to associate documentation for each module, function using docstrings. Docstrings are strings written at the top of the module or at the beginning of a function.

Lets try to document our *num* module by changing the contents of *num.py*

```
"""The num module provides utilties to work on numbers.

Current it provides square and cube.
"""

def square(x):
    """Computes square of a number."""
    return x * x

def cube(x):
    """Computes cube of a number."""
    return x * x
```

The pydoc command will now show us the doumentation nicely formatted.

```
Help on module num:

NAME
    num - The num module provides utilties to work on numbers.

FILE
    /Users/anand/num.py

DESCRIPTION
    Current it provides square and cube.

FUNCTIONS
    cube(x)
        Computes cube of a number.

    square(x)
        Computes square of a number.
```

Under the hood, python stores the documentation as a special field called *__doc__*.

```
>>> import os
>>> print os.getcwd.__doc__
getcwd() -> path

Return a string representing the current working directory.
```

3.1. Standard Library

Python comes with many standard library modules. Lets look at some of the most commonly used ones.

3.1.1. os module

The *os* and *os.path* modules provides functionality to work with files, directories etc.

Problem 1: Write a program to list all files in the given directory.

Problem 2: Write a program *extcount.py* to count number of files for each extension in the given directory. The program should take a directory name as argument and print count and extension for each available file extension.

```
$ python extcount.py src/
14 py
4 txt
1 csv
```

Problem 3: Write a program to list all the files in the given directory along with their length and last modification time. The output should contain one line for each file containing filename, length and modification date separated by tabs.

Hint: see help for `os.stat`.

Problem 4: Write a program to print directory tree. The program should take path of a directory as argument and print all the files in it recursively as a tree.

```
$ python dirtree.py foo
foo
|-- a.txt
|-- b.txt
|-- code
|   |-- a.py
|   |-- b.py
|   |-- docs
|   |   |-- a.txt
|   |   \-- b.txt
|   \-- x.py
\-- z.txt
```

3.1.2. urllib module

The *urllib* module provides functionality to download webpages.

```
>>> import urllib
>>> response = urllib.urlopen("http://python.org/")
>>> print response.headers
Date: Fri, 30 Mar 2012 09:24:55 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Fri, 30 Mar 2012 08:42:25 GMT
ETag: "105800d-4b7b-4bc71d1db9e40"
Accept-Ranges: bytes
Content-Length: 19323
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

>>> response.header['Content-Type']
'text/html'

>>> content = request.read()
```

Problem 5: Write a program wget.py to download a given URL. The program should accept a URL as argument, download it and save it with the basename of the URL. If the URL ends with a /, consider the basename as index.html.

```
$ python wget.py http://docs.python.org/tutorial/interpreter.html
saving http://docs.python.org/tutorial/interpreter.html as interpreter.html.

$ python wget.py http://docs.python.org/tutorial/
saving http://docs.python.org/tutorial/ as index.html.
```

3.1.3. re module

Problem 6: Write a program antihtml.py that takes a URL as argument, downloads the html from web and print it after stripping html tags.

```
$ python antihtml.py index.html
...
The Python interpreter is usually installed as /usr/local/bin/python on
those machines where it is available; putting /usr/local/bin in your
...
```

Problem 7: Write a function *make_slug* that takes a name converts it into a slug. A slug is a string where spaces and special characters are replaced by a hyphen, typically used to create

blog post URL from post title. It should also make sure there are no more than one hyphen in any place and there are no hyphens at the biginning and end of the slug.

```
>>> make_slug("hello world")
'hello-world'
>>> make_slug("hello  world!")
'hello-world'
>>> make_slug(" --hello-  world--")
'hello-world'
```

Problem 8: Write a program *links.py* that takes URL of a webpage as argument and prints all the URLs linked from that webpage.

Problem 9: Write a regular expression to validate a phone number.

3.1.4. json module

Problem 10: Write a program myip.py to print the external IP address of the machine. Use the response from `http://httpbin.org/get` and read the IP address from there. The program should print only the IP address and nothing else.

3.1.5. zipfile module

The *zipfile* module provides interface to read and write zip files.

Here are some examples to demonstate the power of zipfile module.

The following example prints names of all the files in a zip archive.

```
import zipfile
z = zipfile.ZipFile("a.zip")
for name in z.namelist():
    print name
```

The following example prints each file in the zip archive.

```
import zipfile
z = zipfile.ZipFile("a.zip")
for name in z.namelist():
    print
    print "FILE:", name
    print
    print z.read(name)
```

Problem 11: Write a python program *zip.py* to create a zip file. The program should take name of zip file as first argument and files to add as rest of the arguments.

```
$ python zip.py foo.zip file1.txt file2.txt
```

Problem 12: Write a program *mydoc.py* to implement the functionality of pydoc. The program should take the module name as argument and print documentation for the module and each of the functions defined in that module.

```
$ python mydoc.py os
Help on module os:

DESCRIPTION

os - OS routines for Mac, NT, or Posix depending on what system we're on.
...

FUNCTIONS

getcwd()
    ...
```

Hints:

- The *dir* function to get all entries of a module
- The *inspect.isfunction* function can be used to test if given object is a function
- *x.__doc__* gives the docstring for x.
- The *__import__* function can be used to import a module by name

3.2. Installing third-party modules

PyPI, The Python Package Index maintains the list of Python packages available. The third-party module developers usually register at PyPI and uploads their packages there.

The standard way to installing a python module is using *pip* or *easy_install*. Pip is more modern and perferred.

Lets start with installing *easy_install*.

- Download the easy_install install script ez_setup.py.
- Run it using Python.

That will install `easy_install`, the script used to install third-party python packages.

Before installing new packages, lets understand how to manage virtual environments for installing python packages.

Earlier the only way of installing python packages was system wide. When used this way, packages installed for one project can conflict with other and create trouble. So people invented a way to create isolated Python environment to install packages. This tool is called virtualenv.

To install `virtualenv`:

    $ easy_install virtualenv

Installing virtualenv also installs the *pip* command, a better replace for *easy_install*.

Once it is installed, create a new virtual env by running the `virtualenv` command.

```
$ virtualenv testenv
```

Now to switch to that env.

On UNIX/Mac OS X:

```
$ source testenv/bin/activate
```

On Windows:

```
> testenv\Scripts\activate
```

Now the virtualenv *testenv* is activated.

Now all the packages installed will be limited to this virtualenv. Lets try to install a third-party package.

```
$ pip install tablib
```

This installs a third-party library called `tablib`.

The `tablib` library is a small little library to work with tabular data and write csv and Excel files.

Here is a simple example.

```
# create a dataset
data = tablib.Dataset()

# Add rows
data.append(["A", 1])
data.append(["B", 2])
data.append(["C", 3])

# save as csv
with open('test.csv', 'wb') as f:
    f.write(data.csv)

# save as Excel
with open('test.xls', 'wb') as f:
    f.write(data.xls)

# save as Excel 07+
with open('test.xlsx', 'wb') as f:
    f.write(data.xlsx)
```

It is even possible to create multi-sheet excel files.

```
sheet1 = tablib.Dataset()
sheet1.append(["A1", 1])
sheet1.append(["A2", 2])

sheet2 = tablib.Dataset()
sheet2.append(["B1", 1])
sheet2.append(["B2", 2])


book = tablib.Databook([data1, data2])
with open('book.xlsx', 'wb') as f:
    f.write(book.xlsx)
```

Problem 13: Write a program `csv2xls.py` that reads a csv file and exports it as Excel file. The prigram should take two arguments. The name of the csv file to read as first argument and the name of the Excel file to write as the second argument.

Problem 14: Create a new virtualenv and install BeautifulSoup. BeautifulSoup is very good library for parsing HTML. Try using it to extract all HTML links from a webpage.

Read the BeautifulSoup documentation to get started.

Previous Next