Modules: Leveraging Internet Data

Python 3

home (../handouts.html)

Importing Python Modules for Versatility and Power

A Module is Python code (a *code library*) that we can *import* and use in our own code -- to do specific types of tasks.

```
import datetime                         # make datetime (a library module) part of our code

dt = datetime.date.today()              # generate a new date object (dt)
print(dt)                                # prints today's date in YYYY-MM-DD format
dt = dt + datetime.timedelta(days=1)
print(dt)                                # prints tomorrow's date
```

Once a module is imported, its Python code is made available to our code. We can then call specialized functions and use objects to accomplish specialized tasks.

Python's module support is profound and extensive. Modules can do powerful things, like manipulate image or sound files, munge and process huge blocks of data, do statistical modeling and visualization (charts) and much, much, much more.

CSV

The CSV module parses CSV files, splitting the lines for us. We read the CSV object in the same way we would a file object.

```
import csv
fh = open('../python_data/students.txt', 'r')  # second argument: default "read"
reader = csv.reader(fh)

for record in reader:     # loop through each row

    print('id:{};  fname:{}; lname: {}'.format(record[0], record[1], record[2]))
```

This module takes into account more advanced CSV formatting, such as quotation marks (which are used to allow commas within data.)

The second argument to **open()** ('rB') is sometimes necessary when the csv file comes from Excel, which output newlines in the Windows format (**\r\n**), and can confuse the **csv** reader.

Writing is similarly easy:

```
import csv
wfh = open('some.csv', 'w')
writer = csv.writer(wfh)
writer.writerow(['some', 'values', "boy, don't you like long field values?"])
writer.writerows([['a', 'b', 'c'], ['d', 'e', 'f'], ['g', 'h', 'i']])
wfh.close()
```

Python as a web client: the urllib module

A Python program can take the place of a browser, requesting and downloading CSV, HTML pages and other files. Your Python program can work like a web spider (for example visiting every page on a website looking for particular data or compiling data from the site), can visit a page repeatedly to see if it has changed, can visit a page once a day to compile information for that day, etc.

urllib is a full-featured module for making web requests. Although the **requests** module is strongly favored by some for its simplicity, it has not yet been added to the Python builtin distribution.

The **urlopen** method takes a url and returns a file-like object that can be **read()** as a file:

```
import urllib.request
my_url = 'http://www.google.com'
readobj = urllib.request.urlopen(my_url)
text = readobj.read()
print(text)
readobj.close()
```

Alternatively, you can call **readlines()** on the object (keep in mind that many objects that can deliver file-like string output can be read with this same-named method:

```
for line in readobj.readlines():
  print(line)
readobj.close()
```

The text that is downloaded is CSV, HTML, Javascript, and possibly other kinds of data.

TypeError: can't use a string pattern on a bytes-like object
This error may occur with some websites. It indicates that an undecoded unicode response was received.

In order to work with the response as a string, we use the **decode()** method:

```
readobj = readobj.decode('utf-8')
```

SSL Certificate Error
Many websites enable SSL security and require a web request to accept and validate an SSL certificate (certifying the identity of the server). **urllib** by default requires SSL certificate security, but it can be bypassed (keep in mind that this may be a security risk).

```
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

my_url = 'http://www.nytimes.com'
readobj = urllib.request.urlopen(my_url, context=ctx)
```

Encoding Parameters: urllib.requests.urlencode()

When including parameters in our requests, we must *encode* them into our request URL. The **urlencode()** method does this nicely:

```
import urllib.request, urllib.parse

params = urllib.parse.urlencode({'choice1': 'spam and eggs', 'choice2': 'spam, spam, bacon and spam'})
print("encoded query string: ", params)
f = urllib.request.urlopen("http://www.google.com?{}".format(params))
print(f.read())
```

this prints:

```
encoded query string: choice1=spam+and+eggs&choice2=spam%2C+spam%2C+bacon+and+spam

choice1:  spam and eggs<BR>
choice2:  spam, spam, bacon and spam<BR>
```