Computer Science 1 — CSci 1100
Lecture 13 — Data Across the Web

Overview

- Many applications you write may end up using data from external sources.
- Data is accessible on the Web in many different forms.
- In this lecture we will:
  - Talk about how to access sources on the Web, as HTML pages or using APIs
  - Talk about different file formats available on the Web
- Suppose we wanted to write an application that read Tweets from the Web, find relevant images from Twitter and write an HTML page that incorporates both the Tweets and the images.

  Look no more, all the skills you need are in this lecture!

Accessing Static Web pages

- A web source can either be a static HTML page, in that case, we can read it like a file. Usually, such sources have an `http` address and end with `.html`.

```
>>> import urllib
>>> f = urllib.urlopen("http://www.cs.rpi.edu/academics/undergrad/12.html")
```

- An `HTML` file contains information delimited by tags of the form:

```
<html>
   <head>
      <title>HTML example for CSCI-100</title>
   </head>
   <body>
      This is a page about <a href="http://python.org">Python</a>.
      It contains links and other information.
   </body>
</html>
```

Exercise: HTML Parsing

- Write a short piece of code to find the title of a Web page.
- Here is a useful string function to remember:

```
>>> str = "This is an example of a string. It is not very long."
>>> str.find(" is ")
4
```

Example: HTML Parsing

- Python already supports an HTML parser module. But, we will not use it for now.
- Let's find all the links (given by anchor tags) in a given HTML page. For example:

  `<a href="http://python.org">Python</a>`
- HTML allows many variations of this tag, for example: `<a target="top" href="http://python.org">`. But, we will only parse the links that conform to the above form.

```
import urllib
f = urllib.urlopen("http://www.cs.rpi.edu/academics/undergrad/12.html")
file = f.read()
links = []
```

```
print "Links found:",
for link in links:
    print link + ",",
```

Accessing applications on the Web

- The dynamic content of a web source may be accessible through a program called an API. In this case, we may need some additional Python modules to access it.
- Example: flickr photo sharing site

```
import flickrapi
flickr = flickrapi.FlickrAPI(apicode, format='etree')
for photo in flickr.walk(tags = 'sun', tag_mode = 'all', safe_search = '1',\
                            sort = 'interestingness-desc'):
    print photo.get('title')
```

- Example: twitter message exchange

```
import oauth2
import simplejson as sj

token = oauth2.Token(access_token, access_secret)
consumer = oauth2.Consumer(consumer_key, consumer_secret)
```

```
client = oauth2.Client(consumer,token)

url = 'http://search.twitter.com/search.json?q=%s;rpp=100;result_type=recent'
query = 'rensselaer engineers'
resp, content = client.request(url%(query))
```

- BEWARE: a web query like the ones above may return way too many results.

Accessing APIs: Concepts

- *you* are the person running a program to access a source on the Web.
- *flickr* (or *twitter*) is an application that responds to specific calls with specific parameters.

  The application has functions similar to the ones you create that you can call remotely! They generally match what you can do in the site.
- *user* is a user registered to the application (for our purposes is an account you created).
- *apicode* is a string identifying users. Users can request generally at most one API code. You need to supply this everytime you request data from the application.
- *flickrapi* is a Python module that handles the calls to flickr and returns objects Python understands (in this case a simple dictionary).

  Note that for an application, there may be multiple modules with similar but different functionality. We will use the above simple module for now which will allow us to access only the public data.
- Get your *flickrapi* code today! You will need it for Lab 6. Go to:

  `http://www.flickr.com/services/api/misc.api_keys.html`

Flickr Example: Available Functions

- Flickr allows users find images for a given keyword (tag) query. It returns the image information (URL of where it is stored and its title).

```
import flickrapi
apicode = 'some-string-you-will-get-from-flickr'
flickr = flickrapi.FlickrAPI(apicode, format='etree')
i =  1
for photo in flickr.walk(tags = 'sun', \
                            tag_mode = 'all',\
                            safe_search = '1',\
                            sort = 'interestingness-desc'):
    print photo.get('title')
    i += 1
    if i == 5:
        break
```

- Flickr also allows users to retrieve images in a given set (like a folder) specified by the id of the set.

```
import flickrapi
apicode = 'some-string-you-will-get-from-flickr'
flickr = flickrapi.FlickrAPI(apicode, format='etree')
photo_set = flickr.walk_set('72157603856136177', per_page=20)
for photo in photo_set:
    print photo.get('title')
```

- Both functions work similar to a file: you can access items returned one by one with a loop.

FlickrAPI Example: Output Formats

- For each photo, the API returns an object that has the following functions:

```
>>> photo.items()
[('secret', '6988065897'), ('title', 'Over the mountains and the sea.....:)'),
('farm', '4'), ('isprimary', '1'), ('id', '2530880977'), ('server', '3024')]
>>> photo.get('secret')
'6988065897'
>>> photo.get('title')
'Over the mountains and the sea.....:)'
>>> photo.get('farm')
'4'
>>> photo.get('id')
'2530880977'
>>> photo.get('server')
'3024'
```

- You can use these to construct the URL (Web address) of the image:

```
>>> URL = "http://farm%s.static.flickr.com/%s/%s_%s_m.jpg" \
     % (photo.get('farm'), photo.get('server'), \
        photo.get('id'), photo.get('secret'))
>>> URL
http://farm4.static.flickr.com/3024/2530880977_6988065897_m.jpg
```

- What does it all mean? Remember Homework#1: images are stored in computer farms containing many individual computers (servers). This is an address to find a specific machine in a form of many servers. Cut and paste this to a browser, you will see the image.

Reading an image off the Web

- Reading an image on your own directory is easy, you just write the name of the image.

```
im = Image.open(image_file_name)
```

- Reading an image from a URL requires a bit more work. We load it into an object to make it appear as if it was a local file (for now we will not explain this in too much detail):

```
import cStringIO
file = urllib.urlopen(url)
im_f = cStringIO.StringIO(file.read())
im = Image.open(im_f)
```

Example: Accessing Twitter

- Twitter supports extensive functionality through its own python interface.
- You need to create an API key through Twitter, which will provide you with four pieces of information:

```
consumer_key
consumer_secret
access_token
access_token_secret
```

- To get these, you need to register an application at:

  `https://dev.twitter.com/apps/new`

  assuming you already have a twitter account.

- You need to supply all these to connect to Twitter using a module called `oauth2`.

```
import oauth2
token = oauth2.Token(access_token, access_token_secret)
consumer = oauth2.Consumer(consumer_key, consumer_secret)
client = oauth2.Client(consumer,token)
```

Querying Twitter: JSON data format

- Twitter returns data values in a data format called JSON:

```
{"id":337561213,
 "id_str":"337561213",
 "name":"Sibel Adali",
 "screen_name":"sibel_adali",
 "url":"http:\/\/www.cs.rpi.edu\/~sibel",
 "description":"Researcher at Rensselaer Polytechnic Institute (RPI)",
 "followers_count":19,
 "friends_count":36,
 "status":{
      "created_at":"Mon Aug 20 21:34:43 +0000 2012",
      "id":237663980192165888,
      "id_str":"237663980192165888",
      "text":"@jahendler Quick correction: it is graduate students for
       now, but we accept entries from undergraduates as well."}
}
```

- What specific keys are present in a given output is determined by the specific call (we will see in a minute).

- You can load a json object given in a string into a Python object using a module called simplejson:

```
>>> import simplejson as sj
>>> x ='{"id":337561213,"name":{"first":"Sibel","last":"Adali"},"friends":[1,2,3]}'
>>> data = sj.loads(x)
>>> data
{'friends': [1, 2, 3], 'id': 337561213, 'name': {'last': 'Adali', 'first': 'Sibel'}}
```

New Python object type: Dictionaries

- In dictionaries, instead of addressing an item by its location (like in strings and lists), you address them by the field name:

```
>>> data[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1

>>> data['friends']
[1, 2, 3]
```

```
>>> data['id']
337561213
>>> data['name']
{'last': 'Adali', 'first': 'Sibel'}
>>> data['name']['last']
'Adali'
```

- You can add new keys and change the value of existing keys:

```
>>> dic = {}
>>> dic[1] = 'xyz'
>>> dic['name'] = 'joe'
>>> dic
{1: 'xyz', 'name': 'joe'}
>>> dic[1] = 'abc'
>>> dic
{1: 'abc', 'name': 'joe'}
```

Example use of Twitter

- Suppose we want to find the tweets for a given keyword, for example the `mars curiosity rover`. We can just query all recent tweets containing this keyword.

```
import oauth2
import simplejson as sj


consumer_key = 'fill-in'
consumer_secret = 'fill-in'
access_token = 'fill-in'
access_secret = 'fill-in'

token = oauth2.Token(access_token, access_secret)
consumer = oauth2.Consumer(consumer_key, consumer_secret)
client = oauth2.Client(consumer,token)

url = 'http://search.twitter.com/search.json?q=%s;rpp=100;result_type=recent'
query = 'marscuriosity'

data = sj.loads(content)
print 'TWEETS ABOUT YOUR QUERY: ' + query + '\n'
for tweet in data.get('results'):
```

Example Twitter queries to try

- Twitter has extensive functionality through its API.
- You can post tweets, read tweets in your timeline, find friends and followers of specific users, find tweets on a specific topic, find tweets in a specific geographic location, find tweets by a user.
- You can see the type of result returned for a specific query at the API documentation: `https://dev.twitter.com/docs/api/1.1`
- Ex: find tweets for a given query. Read more at:

  `https://dev.twitter.com/docs/api/1.1/get/search/tweets`

- Ex: find friends of a user. Read more at:
  `https://dev.twitter.com/docs/api/1.1/get/friends/ids`

Summary

- You can access data sources by simply reading static files with the help of the `urllib` module.
- Different data sources support additional functions, which can be accessed with modules.
- We learnt two different file types, with their own syntax: HTML and JSON.
- Almost all common file types have parsers, allowing you to convert them into data types available in Python.
- Dictionaries is a new data type, we will more in the next lecture.
- Remember, calls to an outside source like Twitter or Flickr may fail if these applications are busy or we have no Internet connection. We will see later how to check for errors like this.
- Sources like Twitter and Flickr limit you to 1,000 calls per hour as well to make sure you do not overburden them.