

Lecture 7

Writing Programs

Steven Ludtke
N420, sludtke@bcm.edu

urllib

```
import urllib.request
f=urllib.request.urlopen("http://blake.bcm.edu/dl/test.html")
data=f.read()
data=str(data,"utf-8")
print(data)
```

HTML

- <http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>
- Declarative language
- HTML is a type of XML, XHTML obeys XML rules more completely
- Python HTMLParser module
- 'commands' in HTML are denoted by
`<command option=value option=value>text</command>`

HTML

```
<HTML>
```

```
<HEAD><TITLE>My Page</TITLE></HEAD>
```

```
<BODY>
```

```
<H3>Hi Everyone</H3>
```

```
<P>This is really just some test text to  
demonstrate how HTML works. I can do  
interesting things like <i>italicize</i>  
make text <b>bold</b>, or even <b><i>both  
together</i></b>. ta da
```

```
</BODY>
```

```
</HTML>
```

RESTful Servers

- Generally return XML
- Client-Server - Servers store data, clients display data
- Stateless - URL uniquely identifies display
- Cacheable - Pages must declare whether their data is
- www.pdb.org/pdb/software/rest.do

RSS

- Small XML files containing frequently updated information. Link to webpage.
- 2 variants, also Atom.
- Required elements (2.0): title, link, description
- Optional elements: language, copyright, managingeditor, webmaster, pubdate, lastbuilddate, category, generator, docs, cloud, ttl, image, rating, textinput, skiphours, skipdays
- example: http://rss.cnn.com/rss/cnn_topstories.rss

RSS

```
import urllib.request
from xml.etree import ElementTree

f=urllib.request.urlopen("http://rss.cnn.com/rss/cnn_topstories.rss")
data=f.read()
data=str(data,"utf-8")
print(data)

e=ElementTree.fromstring(data)

for el in e.iter():
    print(el)

for txt in e.itertext():
    print(txt)
```

Read/write files

- `handle=open(<filename>,<mode>)`
- Valid modes: `[r|w|a|U][+][b]`
 - `r` - open file for reading
 - `w` - truncate file and open for writing
 - `a` - open file for appending (writing at end of file, platform dependent)
 - `U` - Universal text file support
 - `+` - in addition to basic mode, permit writing
 - `b` - open in binary mode (default is text mode)
- Different platforms do a newline differently:
 - Unix - `'\n'`
 - Old mac - `'\r'`
 - Windows - `'\r\n'`

File Methods

- Binary (application specific) vs. text (ascii) vs. unicode files
- `string=file.read([len])` - Reads whole file (or [len] bytes)
- `string=file.readline()` - Read a single line of text
- `stringlist=file.readlines()` - Read whole file as a list of lines
- `file.write(<string>)` - Write <string> to file (no automatic /n)
- `file.close()` - Close the file (automatic when file object freed)
- `file.flush()` - Write output to file immediately (no buffering)
- `int=file.tell()` - Current location in the file (use binary mode!)
- `file.seek(<loc>)` - Move to a specific position in the file
- `for line in file: print line` - File acts as an iterator for lines
- `sys.stdin, stdout, stderr` - Automatic file handles

Programs you can Run

- Do NOT use a word-processor like MS Word, Pages, etc. You must use a simple 'text editor'. There are built in editors on each platform which will work, but Jupyter has one built in, which may be the simplest approach, unless you prefer another.
- Once you save the file to disk (use a .py extension):
- Just type *python program.py*

-or-

- for Windows, just type *program.py*
- for unix/mac, put:

```
#!/usr/bin/env python
```

on the first line of the file, and type:

```
chmod a+x file.py
```

Then you can just type: *program.py*

- NOTE: on windows, as soon as the program exits, the window showing the output will close. If you put a *input()* at the end of your program, it will wait until you press enter before closing the window so you can see the output.

Arguments

```
myprogram.py file1.txt file2.txt
```

```
python myprogram.py file1.txt file2.txt
```

```
from sys import argv
```

```
len(argv) -> 3
```

```
argv[0] -> myprogram.py
```

```
argv[1] -> file1.txt
```

Note: Can't do this if you launch the program by clicking on it, command-line only!

Standard argument style

myprogram.py file1.txt -Xa --name=abc --option --value 5

- Single dash precedes a list of single-letter options
- Double dash precedes a named option
 - Named options may have a value
 - The separator between the option name and the value may be either '=' or ' '
- Arguments appear in-order without a '-' or '--'
- Arguments may appear before or after options (usually)
- --help or -h will normally provide program help
- when describing options or arguments braces have meaning: <mandatory> or [optional]

argparse - More advanced command parsing

```
import argparse

parser = argparse.ArgumentParser(usage="""This explains what the program
does""")

parser.add_argument('general', type=str, nargs='+', help='general arguments')
parser.add_argument("--name", type=str, help="Name of the
object", default="Fred")
parser.add_argument("--zipcode", type=int, help="Address zip code")
parser.add_argument("--save", action="store_true", help="Save the results")
args = parser.parse_args()

print(args)
```