Making HTTP Requests

- Using Plain Old Python
  - Requests
  - Manipulating URLs
  - Encoding parameters
- The (amazing) `requests` library
  - HTTP Verbs
  - Headers
  - Proxies
- Parsing HTML with Beautiful Soup
  - CSS selectors
- Browser emulation with Selenium

Using Plain Old Python

It's possible to do in plain Python, but it's quite verbose:

In [3]:
```
import httplib
conn = httplib.HTTPSConnection("www.python.org")
conn.request("GET", "/")
r1 = conn.getresponse()
print r1.status, r1.reason
html = r1.read()
print html[:100]
```

```
200 OK
<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <!
[endif]-->
<!-
```

The `requests` library

One of the single-best libraries for Python, `requests` (http://docs.python-requests.org/en/master/) makes it quite easy to make all sorts of HTTP requests, and do quite advanced things such as:

- GET, PUT, POST, DELETE, etc
- Set headers
- Use proxies
- Use HTTP Authentication
- Use cookies / sessions
- File uploads
- Streaming downloads

...and the list goes on and on.

In [7]:
```
import requests

# make a GET request
r = requests.get("http://google.com/")
print "=> Google:", r.content[:100]

# make a POST request
r = requests.post("http://httpbin.org/post", params={
    "first" : 1,
    "second" : "two",
})

print "=> HTTPBin:", r.content
```

```
=> Google: <!doctype html><html itemscope="" itemtype="http://schema.or
g/WebPage" lang="en"><head><meta content
=> HTTPBin: {
  "args": {
    "first": "1",
    "second": "two"
  },
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Content-Length": "0",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.9.1"
  },
  "json": null,
  "origin": "50.1.125.102",
  "url": "http://httpbin.org/post?second=two&first=1"
}
```

More Advanced `requests`

In [7]:
```
import requests

# we won't use this (fake), but this is the format
proxy = {"http": "http://username:p3ssw0rd@10.10.1.10:3128"}

url = 'http://api.openweathermap.org/data/2.5/weather'
params = {"q" : "London,uk", "APPID" : "eb7f2b3b74d14e1e3ca82564c6d3585
4"}
headers = {
    "User-Agent" :
        "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)" +
        "AppleWebKit/537.36 (KHTML, like Gecko)" +
        "Chrome/57.0.2987.133 Safari/537.36"}

# make the request
r = requests.get(url, params=params, headers=headers)
print "=> JSON:",r.content, '\n'
print "=> Response object:", r.__dict__
```

```
=> JSON: {"coord":{"lon":-0.13,"lat":51.51},"weather":[{"id":721,"mai
n":"Haze","description":"haze","icon":"50d"}],"base":"stations","main":
{"temp":291.07,"pressure":1021,"humidity":45,"temp_min":289.15,"temp_ma
x":293.15},"visibility":10000,"wind":{"speed":3.6,"deg":170},"clouds":
{"all":0},"dt":1491672000,"sys":{"type":1,"id":5091,"message":0.0057,"c
ountry":"GB","sunrise":1491628696,"sunset":1491677220},"id":2643743,"na
me":"London","cod":200}

=> Response object: {'cookies': <RequestsCookieJar[]>, '_content': '{"c
oord":{"lon":-0.13,"lat":51.51},"weather":[{"id":721,"main":"Haze","des
cription":"haze","icon":"50d"}],"base":"stations","main":{"temp":291.0
7,"pressure":1021,"humidity":45,"temp_min":289.15,"temp_max":293.15},"v
isibility":10000,"wind":{"speed":3.6,"deg":170},"clouds":{"all":0},"d
t":1491672000,"sys":{"type":1,"id":5091,"message":0.0057,"country":"G
B","sunrise":1491628696,"sunset":1491677220},"id":2643743,"name":"Londo
n","cod":200}', 'headers': {'X-Cache-Key': '/data/2.5/weather?APPID=eb7
f2b3b74d14e1e3ca82564c6d35854&q=london%2cuk', 'Content-Length': '437',
 'Server': 'openresty', 'Connection': 'keep-alive', 'Access-Control-All
ow-Credentials': 'true', 'Date': 'Sat, 08 Apr 2017 18:00:29 GMT', 'Acce
ss-Control-Allow-Origin': '*', 'Access-Control-Allow-Methods': 'GET, PO
ST', 'Content-Type': 'application/json; charset=utf-8'}, 'url': u'htt
p://api.openweathermap.org/data/2.5/weather?q=London%2Cuk&APPID=eb7f2b3
b74d14e1e3ca82564c6d35854', 'status_code': 200, '_content_consumed': Tr
ue, 'encoding': 'utf-8', 'request': <PreparedRequest [GET]>, 'connectio
n': <requests.adapters.HTTPAdapter object at 0x10a342610>, 'elapsed': d
atetime.timedelta(0, 0, 207079), 'raw': <requests.packages.urllib3.resp
onse.HTTPResponse object at 0x10a3428d0>, 'reason': 'OK', 'history':
 []}
```

Parsing HTML with Beautiful Soup

We certainly don't want to use regex (http://stackoverflow.com/a/1732454/). So we'll use a better tool.

```
$ pip install beautifulsoup4
```

```
In [9]: import requests
        from bs4 import BeautifulSoup

        r = requests.get("https://en.wikipedia.org/wiki/Donald_Trump")
        soup = BeautifulSoup(r.content, "html.parser")

        # find all the <h2> tags
        soup.find_all('h2')
```

```
Out[9]: [<h2>Contents</h2>,
         <h2><span class="mw-headline" id="Early_life">Early life</span></h2>,
         <h2><span class="mw-headline" id="Real_estate_business">Real estate bu
        siness</span></h2>,
         <h2><span class="mw-headline" id="Side_ventures">Side ventures</span>
        </h2>,
         <h2><span class="mw-headline" id="Media_career">Media career</span></h
        2>,
         <h2><span class="mw-headline" id="Political_career_up_to_2015">Politic
        al career up to 2015</span></h2>,
         <h2><span class="mw-headline" id="2016_presidential_campaign">2016 pre
        sidential campaign</span></h2>,
         <h2><span class="mw-headline" id="Presidency">Presidency</span></h2>,
         <h2><span class="mw-headline" id="Electoral_history">Electoral history
        </span></h2>,
         <h2><span class="mw-headline" id="Personal_life">Personal life</span>
        </h2>,
         <h2><span class="mw-headline" id="Awards.2C_honors.2C_and_distinction
        s">Awards, honors, and distinctions</span></h2>,
         <h2><span class="mw-headline" id="See_also">See also</span></h2>,
         <h2><span class="mw-headline" id="Notes">Notes</span></h2>,
         <h2><span class="mw-headline" id="References">References</span></h2>,
         <h2><span class="mw-headline" id="Bibliography">Bibliography</span></h
        2>,
         <h2><span class="mw-headline" id="External_links">External links</span
        ></h2>,
         <h2>Navigation menu</h2>]
```

Using Selectors to find elements

Often times, there's a lot of HTML to sift through, and we'd rather be a bit more precise. BeautifulSoup allows us to do this by using CSS selectors as well as the tag type:

```
In [ ]: # find just internal Wikipedia links
        soup.find_all("a", attrs={"class", "internal"})
```

Browser Emulation with Selenium

Sometimes you might want to take things a bit farther and actually emulate the browser, Javascript, image rendering, and all. This is sometimes for sneakier purposes like web scraping undetected or more mundane like doing regression testing on browser apps. Either way, Selenium is a good tool.

```
$ pip install selenium
```

In [ ]:
```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("http://www.imdb.com/")

# find input box and type
query_input = driver.find_element_by_css_selector("#navbar-query")
query_input.send_keys("godzilla")

# find search button and click
search_button = driver.find_element_by_css_selector("#navbar-submit-butt
on")
search_button.click()

driver.quit()  # exit browser
```

Lab: Coding Excercises

Fill in the method definitions in the file `excercises/http.py`.

Make sure you can pass tests with:

```
$ py.test tests/test_http.py::HTTPExcercises::<function_name>  # test single
 function
$ py.test tests/test_http.py::HTTPExcercises                   # test all at
 once
```

Wrap-Up

- Using Plain Old Python
  - Requests
  - Manipulating URLs
  - Encoding parameters
- The (amazing) `requests` library
  - HTTP Verbs
  - Headers
  - Proxies
- Browser emulation with Selenium