# Experiment 2

**Class: F1503023**

**Name: Wang Jialu**

**ID: 515030910558**

# HTTP Protocol

## What is HTTP?

The **Hypertext Transfer Protocol** (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

To send a HTTP request, we need such things:

- Target URL
- Headers
- Query String

## HTTP Methods: GET vs. POST

The two most used HTTP methods are: GET and POST.

### The GET Method

**Note that the query string (name/value pairs) is sent in the URL of a GET request:**

```
/test/demo_form.asp?name1=value1&name2=value2
```

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

### The POST Method

**Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:**

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

# Crawler

Here I will show how to write a crawler step by step.

## First step: Establish connections

I use urllib to establish connections. Also there're other alternatives but TA don't persuade.

```
def get_page(page):
    content = ''
    try:
        req = urllib2.urlopen(page)
    except:
        pass
    else:
        content = req.read()
    return content
```

Consedering that some links can't reach, I use 'try' before open the url. I don't deal with the Exception and don't mind it.

## Second step: Store content and fetch urls

At first step I get all the content in a website. So I need to store it.

```
def add_page_to_folder(page, content):
    index_filename = 'index.txt'
    folder = 'html'
    filename = valid_filename(page)
    index = open(index_filename, 'a')
    index.write(page.encode('ascii', 'ignore') + '\t' + filename + '\n')
    index.close()
    if not os.path.exists(folder):
        os.mkdir(folder)
    f = open(os.path.join(folder, filename), 'w')
    f.write(content)
    f.close
```

To get all the links I use BeautifulSoup and regex to match url. If you only use regex, some websites are not so correct that you may match something wrong. But BeautifulSoup may be a little slow, and lxml maybe a better choice.

```
def get_all_links(content, page):
    links = []
    soup = BeautifulSoup(content)
    for a in soup.findAll('a', {'href': re.compile('^http|^/')}):
        url = a.get('href')
        if url[0] == '/':
            url = urlparse.urljoin(page, url)
        links.append(url)
    return links
```

Here I consider the url may be relative path so I deal with that.

## Two methods: BFS and DFS

BFS will search along the breadth of a tree. Here is the algorithm, which unions a queue to another.

```
def union_bfs(a,b):
    for ele in b:
        if ele not in a:
            a.insert(0, ele)
```

Oppositely, DFS search as deep as possible.

```
def union_dfs(a,b):
    for e in b:
        if e not in a:
            a.append(e)
```

The algorithm above unions a list to another.

# Exercise

## Problem 1

> 使用自己的账号模拟登陆BBS后，修改个人说明档（修改bbs*set*sample.py）

BBS is too low that you just need to post such form to login:

```
{
    'id': id,
    'pw': pw,
    'submit':'login'
}
```

The difficulty is that you need use the cookies to post your bbsplan. Here's the code to take advantage of cookies.

```
    cj = cookielib.CookieJar()
    opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
    urllib2.install_opener(opener)
```

And you can change your bbsplan easily.

```
    postdata = urllib.urlencode({
        'type': 'update',
        'text': text
        })
    req = urllib2.Request(url='https://bbs.sjtu.edu.cn/bbsplan', data=postdata)
    response = urllib2.urlopen(req)
```

## Problem 2

> 修改crawler*sample.py中的union*bfs函数，完成BFS搜索

```
def union_bfs(a,b):
    for ele in b:
        if ele not in a:
            a.insert(0, ele)
```

Aren't it easy?

## Problem 3

> 修改crawler_sample.py中的crawl函数，返回图的结构

Most code have been done. So I just add several lines:

```
def crawl(seed, method):
    tocrawl = [seed]
    crawled = []
    graph = {}
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            outlinks = get_all_links(content)
            globals()['union_%s' % method](tocrawl, outlinks)
            if page in g.keys:
                graph[page] = g[page]
            else:
                graph[page] = []
    return graph, crawled
```

Attention that you should check if page is in list *g* before add it to *graph*.

## Problem 4

Based on problems before, I just complete it quickly!

Some links might be not able to open, so I use *try* to acquire the content.

```
def get_page(page):
    content = ''
    try:
        req = urllib2.urlopen(page)
    except:
        pass
    else:
        content = req.read()
    return content
```

And I use BeautifulSoup and regex to match url.

```
f get_all_links(content, page):
    links = []
    soup = BeautifulSoup(content)
    for a in soup.findAll('a', {'href': re.compile('^http|^/')}):
        url = a.get('href')
        if url[0] == '/':
            url = urlparse.urljoin(page, url)
        links.append(url)
    return links
```

You can get all the code in the attachment.