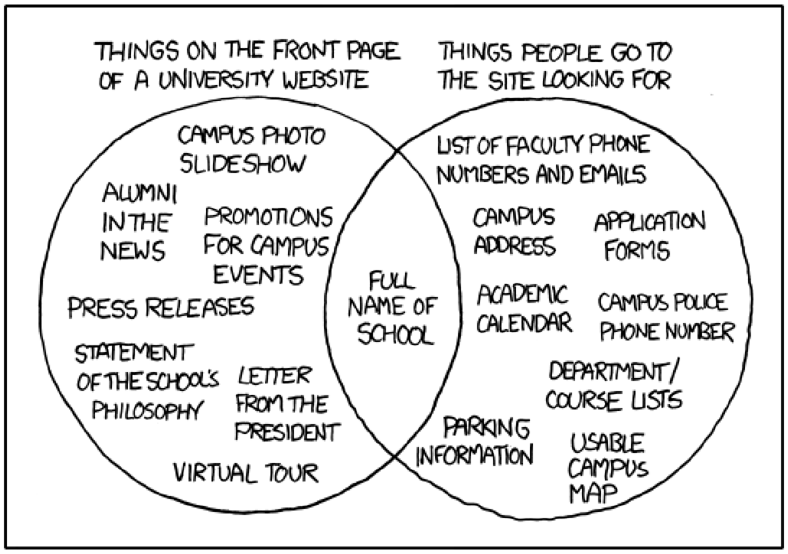
CS51A - Assignment 11

Due Tuesday, November 25, at 11:59pm


THINGS ON THE FRONT PAGE OF A UNIVERSITY WEBSITE
THINGS PEOPLE GO TO THE SITE LOOKING FOR
CAMPUS PHOTO SLIDESHOW
ALUMNI IN THE NEWS
PROMOTIONS FOR CAMPUS EVENTS
PRESS RELEASES
STATEMENT OF THE SCHOOL'S PHILOSOPHY
LETTER FROM THE PRESIDENT
VIRTUAL TOUR
FULL NAME OF SCHOOL
LIST OF FACULTY PHONE NUMBERS AND EMAILS
CAMPUS ADDRESS
APPLICATION FORMS
ACADEMIC CALENDAR
CAMPUS POLICE PHONE NUMBER
DEPARTMENT/ COURSE LISTS
PARKING INFORMATION
USABLE CAMPUS MAP

`https://xkcd.com/773/`

For our last (yay!?) assignment, we're going to be writing a basic webcrawler that finds `pomona.edu` webpages. For this assignment in particular, make sure to develop your program incrementally and test each of your functions carefully before moving on. We will be making live queries to webservers and it's important that we are respectful of those resources.

As always, read through the entire handout first before starting.

Starter

Create a directory called `assignment11` and download and unzip the starter file into this directory:

`http://www.cs.pomona.edu/classes/cs51a/assignments/assign11-starter.zip`

The starter includes a file called `stackqueue.py` which contains implementations of a stack and queue that we saw in class. To use these in your program, create a file called `assign11.py` and include the following line at the top of the file with your other import statements:

```
from stackqueue import *
```

Overview

Your webcrawler will start at a `pomona.edu` webpage and will perform a search (BFS or DFS) following the hyperlinks. The search will utilize a `to_visit` structure (either a stack or a queue) to keep track of what webpages it has found, but has not visited. To avoid visiting a page that we've already visited, we'll also need to keep track of those pages we've already processed. The high-level algorithm is as follows:

– Get a URL from the `to_visit` data structure (stack or queue).

– Check to make sure it's not a webpage we've visited already.

– Download the webpage associated with the URL and convert it to text.

– Identify all of the links on the webpage from the text.

– Filter that list to only include those that are `pomone.edu` links.

– Add those links to `to_visit`.

– Repeat until we've found a fixed number of links.

Crawling to the finish line

1. [**1 point**] Write a function called `is_valid_pomona_url` that takes as input a string representing a URL and returns `True` if that website is a `pomona.edu` website and `False` otherwise.

```
>>> is_valid_pomona_url("http://www.google.com")
False
>>> is_valid_pomona_url("http://www.pomona.edu")
True
>>> is_valid_pomona_url("http://www.cs.pomona.edu")
True
>>> is_valid_pomona_url("https://cs.pomona.edu/~dkauchak/")
True
```

2. [**1 point**] Write a function called `is_full_url` that takes as input a string representing a URL and returns `True` if that website is a valid full website address and not a relative link. Full websites should start with `http`. In addition, a valid url shouldn't have any spaces.

3. [**5 points**] Write a function called `get_all_urls` that takes as input a string representing a URL and returns a list containing all *full* web links that can be found on that page.

   Specifications:

   - The function should be able to handle `http` and `https` webpages as input.

- Use "ISO-8859-1" to decode the webpages.
- Links should be found using the `a href` html tag and you can assume that the linked URL will be surrounded in double quotes.
- To make this function more robust, your function should be able to handle two common problems that can occur when fetching a webpage: 1) the server doesn't exist and 2) the server exists, but the page you requested doesn't exist. Both of these cases will raise an exception when `urlopen` is called.

  The names for these two exceptions are `urllib.error.URLError` and `urllib.error.HTTPError`. To include these exceptions, import `urllib` at the top of your file and then include a `try` block with two `except` cases, one for each exception. If an exception occurs, your function should print out "Ignoring: " followed by the URL and return an empty list.

Here are a few test cases you can try out:

```
 >>> get_all_urls("http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html")
 ['http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test1.html',
  'http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test2.html',
  'http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test3.html']

 >>> get_all_urls("http://www.cs.pomona.edu/classes/cs51a/")
['http://www.cs.pomona.edu/classes/cs51a/',
'https://runestone.academy/runestone/books/published/thinkcspy/index.html',
'https://docs.python.org/3/tutorial/',
'https://docs.python.org/3/',
'https://docs.python.org/3/library/']


 >>> len(get_all_urls("https://cs.pomona.edu/~dkauchak"))
 69
```

4. [**2 points**] Write a function called `filter_pomona_urls` that takes as input a list of URLs and returns a new list of URL containing *only* the `pomona.edu` URLs from the input list.

```
 >>> filter_pomona_urls(get_all_urls("http://www.cs.pomona.edu/classes/cs51a/"))
['http://www.cs.pomona.edu/classes/cs51a/']
```

5. [**5 points**] Write a function called `crawl_pomona` that has three parameters: a URL to start the crawl at, a `to_visit` structure (either a `Stack` or a `Queue`), and a number representing the maximum number of URLs to crawl. The function should return a list of all of the `pomona.edu` websites it can visit starting the crawl from the starting URL.

   Specifications:

   - Your code should follow the general search approach using `to_visit` described in class (and outlined above).

- You should keep track of the websites you've visited (i.e., removed from `to_visit` as a `set` and you should make sure never to visit a website twice. If you find a website you've already visited, just ignore it. Note that "visiting" a website is only done when it is removed from the `to_visit` structure *not* when it's added.
- Each time you visit a website, your function should print out "Crawling: " followed by the URL.
- Your crawler should be a *nice* crawler. To accomplish this, import the `time` module and use the `time.sleep` function to sleep for 0.1 seconds after you visit each URL.
- Your function should finish when either 1) you run out of links to visit or 2) if you hit the maximum number of URLs to crawl (third parameter).

Hints:

- Use the general search code that we looked at in class that uses either a stack or a queue to help guide writing this function. It will be a bit different, but will have many similarities.
- I've put together some very basic test pages to help you in debugging. The first is at:

  `http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html`

  This pages links to three more pages (test1, test2 and test3). test2 in turn links to two more pages test4 and test5. test2, test3, test4 and test5 are terminal and don't have any links.

  The second is a set of two pages that link to each other. This will help you in checking to make sure you're not visiting the same pages over and over again. The page:

  `http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/testloop.html`

  links to a second page, testloop1, which links back to the original testloop page.
- As always, develop incrementally and test as you go. For example, here's one way that you might do this:
  (a) Write the function without a while loop and just make sure it processes the first page correctly, i.e. extracts only the `pomona.edu` URLs.
  (b) Add in the loop and test on the simple test example above. There are no loops in the link structure of this webpage, so you won't need to worry about revisiting a webpage multiple times. Check that you function both terminates when it runs out of URLs to visit and when the max number if hit.
  (c) Add in functionality to keep track of where you've visited already. You can use the testloop examples below to help you with this.

Here are a few examples runs on the test cases:

```
>>> crawl_pomona("http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html",
    Queue(), 20)
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test1.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test2.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test3.html
```

```
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test4.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test5.html
['http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html',
 'http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test1.html', 'http://www.cs.pomona.edu/clas
 'http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test4.html', 'http://www.cs.pomona.edu/clas


>>> crawl_pomona("http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html",
    Stack(), 3)
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test3.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test2.html
['http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test.html',
 'http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/test3.html',  'http://www.cs.pomona.edu/cla

>>> crawl_pomona("http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/testloop.html",
    Stack(), 5)
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/testloop.html
Crawling: http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/testloop1.html
['http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/testloop.html',
 'http://www.cs.pomona.edu/classes/cs51a/assignments/assign11/testloop1.html']
```

6. [**2 points**] Write a function called `write_pomona_urls` that has four parameters: a starting URL, a to visit structure (`Stack` or `Queue`), the maximum number to crawl, and an output filename. The function should crawl pomona webpages (using your crawl function) and then write the results on the output file.

Experiments

Now that you have a working web crawler, experiment with how using DFS vs. BFS affects the order in which the pages are visited. In particular, run your `write_pomona_urls` function starting at `https://www.pomona.edu/academics/departments/computer-science/faculty-staff` using both a stack and a queue and save the first 100 webpages it finds. Include a paragraph (3-4 sentences) as a triple quoted string at the end of your file that includes two things: 1) your observations about how the results differ and 2) if we were only going to select 100 URLs, which search would be better to use and why.

One note: Since we're staying within `pomona.edu`, the crawl should be pretty stable, however, we are dealing with webpages and the content can be changed and other unexpected things can happen. If you find that your webcrawler is getting stuck on a particular page for a long time, you can add a *timeout* parameter to `urlopen` that tells it to stop trying after a certain number of seconds, e.g.:

```
page = urlopen(link, timeout=2)
```

When you're done

You should have a single file, `assign11.py` with the functions above and a triple quoted string at the end with your experiment observations.

Make sure that the file is properly commented:

- You should have comments at the very beginning of the file stating your name, course, assignment number and the date.
- Each function should have an appropriate docstring.
- Include other miscellaneous comments to make things clear.

Submit your .py file online using the course submission mechanism.

Grading

	points
`is_valid_pomona_url`	1
`is_full_url`	1
`get_all_urls`	5
`filter_pomona_urls`	2
`crawl_pomona`	5
`write_pomona_urls`	2
Experiments	2
comments/style	3
total	21