Homework 6: Fall 2020

Fill in your name

In [136]:
```
first_name = "Shariq"
last_name = "Jamil"

assert(len(first_name) != 0)
assert(len(last_name)  != 0)
```

Problem 1: Mailman

Turn an e-mail address into a list of components

We address letters and and e-mail backwards. When the post office gets a letter, they need to read from the bottom up to decide where to send it next

```
Stephen Dedalus
    Class of Elements
    Clongowes Wood College
    Sallins
    Country Kildare
    Ireland
```

Internet addresses such as 'jparker@word.std.com' work the same way.

Write a function that takes a string holding an e-mail address and returns a list with two items: the username, followed by a list of the steps we will need to take to route the mail. In the case above, you would return

```
['jparker', ['com', 'std', 'world']]
```

Hint: Use the string method split() twice.

In [137]:
```
# Takes a string and returns a list
def parse_email_address(s):
    # error if not a valid email address
    if '@' not in s:
        raise ValueError('Not a valid email address')

    # split email address at '@' sign
    email_list = s.split('@')
    # first half is username
    username = email_list[0]
    # the rest is what comes after @
    suffix_list = email_list[1].split('.')

    suffix = []

    # iterate through suffix and store in reverse order
    count = len(suffix_list)-1
    while count != -1:
        suffix.append(suffix_list[count])
        count = count - 1

    # create return object
    email = []
    email.append(username)
    email.append(suffix)
    return email
```

Test cases for Mailman

In [138]:
```
def mailman_test():
    assert(parse_email_address('jdp@world.std.com') == ['jdp', ['com',
'std', 'world']])

    return('Pass')

mailman_test()
```

Out[138]: 'Pass'

Problem 2: Parentheses

Decide if a string contains valid nested parentheses

You are given a string consisting only of parentheses - (,), {, }, [, and]. Write a Boolean function is_valid_parens() that takes a string and decides if it consists of valid nested parenthesis.

Hint: Your function should take open parentheses, such as '(', and 'push it on a stack' and should take closing parentheses, and pop the stack and compare. If the close parenthesis doesn't match the open parenthesis on top of the stack, the string is invalid. If the stack is empty too soon, or is not empty when you finish the string, the string is invalid.

You can read about stacks here:

https://en.wikipedia.org/wiki/Stack_(abstract_data_type (https://en.wikipedia.org/wiki/Stack_(abstract_data_type))

Implement your stack with a list, pushing and poping the final element.

In [139]:
```
# Takes a string, and returns a Boolean
#  '{()[{}]}' is valid:    return True
#  '{()[{}}' is not:       return False
def is_valid_parens(s):
    # do not process if array is empty
    if(len(s) == 0):
        return True

    # create dicts that store open/close parens
    open_parens = {}
    open_parens['['] = ']'
    open_parens['('] = ')'
    open_parens['{'] = '}'

    close_parens = {}
    close_parens[']'] = '['
    close_parens[')'] = '('
    close_parens['}'] = '{'

    # array used to process parens
    parens = []

    # iterate through string
    counter = 0
    while counter < len(s):
        chr = s[counter]
        # if there are closed parens to be found
        if(len(parens) > 0):
            # and the char is an open paren
            if(chr in open_parens):
                # add it to processing list
                parens.append(chr)
            else:
                # it is not an open paren char, has to match last char
in processing list
                # the char should be the matching open paren
                if (chr in close_parens and parens[len(parens)-1] == c
lose_parens[chr]):
                    # match found, remove open paren
                    parens.pop()
                else:
                    # match not found, failure
                    return False
        else:
            # open paren, add for processing and move on
            parens.append(chr)
        counter+= 1
    # open parens without matching close parens found, failure
    if(len(parens)>0):
        return False
    # there is a closed paren for each open paren, pass!
    return True
```

Test case for is_valid_parens()

In [140]:
```
def test_parens():
    assert(is_valid_parens(""))
    assert(is_valid_parens("[]"))
    assert(is_valid_parens('{()[{}]}'))
    assert(is_valid_parens("{}"))
    assert(is_valid_parens("{[]}"))
    assert(is_valid_parens("{}[]"))
    assert(is_valid_parens("([{}({}[])])"))

    assert not is_valid_parens('{()[{}}]'), 'Interlaced parentheses'
    assert not is_valid_parens("[["), "Unmatched opens"
    assert not is_valid_parens("}{"), "Unmatched close"
    assert not is_valid_parens("{]"), "Missmatched parentheses"
    assert not is_valid_parens("{[])"), "Missmatched parentheses"
    assert not is_valid_parens("{()][]}"), "Missmatched parentheses"
    assert not is_valid_parens("([{])"), "Missmatched parentheses"
    assert not is_valid_parens("[({]})"), "Missmatched parentheses"

    return 'Pass'

test_parens()
```

Out[140]:
```
'Pass'
```

Problem 3: Solitary Words

A word is *solitary* if no letter appears more than once.

Thus 'once' is solitary. 'Solitary' is solitary. 'Pop' is not solitary, as there are two p's.

Write a function that takes a string and returns a Boolean telling us if the string is solitary.

```
def solitary(word: str) -> bool:
```

Hint:

Review Downey's discussion of Dictionary as a Collection of Counters.

Hint:

You can also use a Set to keep track of the letters in a word

Hint:

You can solve this without Sets or Dictionaries. Please don't. This is the only problem on this set that uses these new ideas.

In [141]:
```
def solitary(word):
    # dictionary for tracking word usages
    dict = {}
    for chr in word:
        # check to see if char has been seen already
        if(chr.lower() in dict):
            # we have seen this char before, fail
            return False
        else:
            # new char, add to dict
            dict[chr.lower()] = 1
    # no repeats found, pass
    return True
```

Unit Test cases for solitary()

In [142]:
```
def test_solitary():
    assert solitary('abcd')
    assert not solitary('aa'), "Two a's"
    assert solitary('solitary')
    assert not solitary('Pop'), "Two p's"
    assert not solitary("eleven"), "Three e's"
    assert solitary("subdermatoglyphic")

    print('Success!')

test_solitary()
```

```
Success!
```

Problem 4: Find Large Files

Write a function that takes a directory and a size in bytes, and returns a list of files in the directory or below that are larger than the size.

For example, you can use this function to look for files larger than 1 Meg below your Home directory.

You will find a Python function that gives you the size of a file in the os.path library:

https://pymotw.com/3/os.path/ (https://pymotw.com/3/os.path/)

In [143]:
```
import os
import sys

# global var to track large files
large_files = []

# Using a modified version of Downey's walk function
def walk(dirname: str, filesize):
    # Walk over the files in this directory
    for name in os.listdir(dirname):
        # Construct a full path
        path = os.path.join(dirname, name)
        # if a file is found
        if os.path.isfile(path):
            # if file size is greater than given filesize
            if(os.path.getsize(path) > filesize):
                # add to list
                large_files.append(path)
        # traverse directory
        else:
            walk(path, filesize)

def find_large_files(dirname, filesize):
    # call walk function to get list of files larger than the given si
ze
    walk(dirname, filesize)
    return large_files
```

Show your program in action

Give the parameters and show the results for your program

I looked for files larger than a Megabyte found below the directory one step up.

In [144]:
```
lst = find_large_files('..', 1048576)
print(len(lst))

for path in lst:
    print(path)
```

```
4
..\.ipynb_checkpoints\cs109a_hw1-checkpoint.ipynb
..\cs109a_hw1.ipynb
..\Homework 4\words.txt
..\Homework 5\words.txt
```

Problem 5:

The following stand-alone program takes a url from the command line, reads the contents of a webpage, and prints it.

Modify the program to take a filename as a second parameter and save the contents of the webpage in a text file.

```
python save_url.py 'http://www.python.org/' pythonpage.txt
```

would save the contents of the webpage in the text file pythonpage.txt.

You may want to review the mycopy.py program from day 4 which takes two parameters and copies the contents of the first file to the second.

Use this and an editor to find the copyright notice on the following websites.

```
website = 'http://www.python.org/'
    website = 'https://www.extension.harvard.edu'
    website = 'http://en.wikipedia.org/wiki/Python'

    website = Your piazza link: mine looks something like this:
            https://piazza.com/class/myxlplyxmyxlplyx?cid=194
```

You will need to remove the last bit from your piazza link that specifies the cid: '?cid=194'. In my case, this would leave https://piazza.com/class/myxlplyxmyxlplyx (https://piazza.com/class/myxlplyxmyxlplyx)

This problem gives you a chance to examine webpages, and shows how different website creators deal with a common problem, presenting a copyright. You will see that writing a program to extract the copyright from different websites would be difficult.

For example, here is the copyright notice for the New York Times, https://www.nytimes.com (https://www.nytimes.com). I have introduced whitespace to help visualize the element.

```
<li data-testid="copyright">
    <a class="css-jq1cx6" href="https://help.nytimes.com/hc/en-us/articles
/115014792127-Copyright-notice">©
        <span>2020</span>
        <span>The New York Times Company</span>
    </a>
</li>
```

One alternative way to view the source for a website is through your browser. For example, in Chrome you can use View/Developer/View Source

In [146]:
```
# Modified given read_url.py
# by Jeff Parker
#
# This func takes in a website URL and a textfile name
# as strings. It then writes the contents of the webpage
# at the URL to a file. The file will use the name passed
# in as the second arg
#
# Usage:
#     python save_contents.py <website> <textfile>

import urllib.request
import sys

def save_contents(website, textfile):
    "Saves the contents of this webpage as a file"
    try:
        # Counter to keep track of lines, just for reference
        counter = 0
        with urllib.request.urlopen(website) as f:
            text = f.read().decode('utf-8')

            # Break the page into lines
            text = text.split('\n')
            # Open the dest file
            with open(textfile,'w',encoding="utf-8") as fout:
                # Grab each line
                for line in text:
                    # Write to file
                    fout.write(line)
                    counter += 1
        # return number of lines processed
        return counter

    except urllib.error.URLError as e:
        print(e.reason)
        return []
```

Include your program below

In [147]:
```
import urllib.request
import sys

# save_url.py
#
# This function takes in a web URL and a textfile name.
# It calls save_contents to perform the actual work.
# As a result of the call, the contents at <website> URL
# will be stored in <textfile>
#
# Usage:
#     python save_url.py <website> <textfile>
def save_url(website, textfile):
    num_lines_processed = save_contents(website, textfile)
    print(num_lines_processed)
```

Show the webpage elements holding the copyright information for each website

Copyright notice for http://www.python.org/ (http://www.python.org/)

```
<div class="copyright">
    <p>
        <small>
            <span class="pre">Copyright &copy;2001-2020.</span> 
            <span class="pre"><a href="/psf-landing/">Python Software Foun
dation</a></span> 
            <span class="pre"><a href="/about/legal/">Legal Statements</a>
</span> 
            <span class="pre"><a href="/privacy/">Privacy Policy</a></span
> 
            <span class="pre"><a href="/psf/sponsorship/sponsors/#heroku">
Powered by Heroku</a></span>
        </small>
    </p>
</div>
```

Copyright notice for https://www.extension.harvard.edu (https://www.extension.harvard.edu)

```
<p>Copyright &copy;2020 President and Fellows of Harvard College</p>
```

Copyright notice for http://en.wikipedia.org/wiki/Python (http://en.wikipedia.org/wiki/Python)

```
<li id="footer-info-copyright">
    Text is available under the <a rel="license" href="//en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License">Creative Commons Attribution-ShareAlike License</a><a rel="license" href="//creativecommons.org/licenses/by-sa/3.0/" style="display:none;"></a>;additional terms may apply.  By using this site, you agree to the <a href="//foundation.wikimedia.org/wiki/Terms_of_Use">Terms of Use</a> and <a href="//foundation.wikimedia.org/wiki/Privacy_policy">Privacy Policy</a>. Wikipedia® is a registered trademark of the <a href="//www.wikimediafoundation.org/">Wikimedia Foundation, Inc.</a>, a non-profit organization.
</li>
```

Copyright notice for Piazza

```
# save_url('https://piazza.com/class/kckxsbcoz20gd','piazza.txt')
<li>
    <a class="log_click" data-log-click="footer_legal_copyright" href="/legal/copyright">Copyright Policy</a>
</li>

<div id="CopyrightInfo">
    Copyright &#169; 2020<br/>Piazza Technologies<br/>All Rights Reserved
</div>
```

Post Mortem

How long did it take you to solve this problem set?

Did anything confuse you or cause difficulty?

In [135]:
```
# Enter your thoughts
# It took me about 6 hours. A joy as always.
```