CS 210 – Introduction to Programming

PEX 2 – Random Picker

Due: 2300, Lesson 21; Tuesday, 14 October (M-Day) / Wednesday, 15 October (T-Day)

Help Policy:

AUTHORIZED RESOURCES: Any, except another cadet's program.

NOTE:

- Never copy another person's work and submit it as your own.
- Do not jointly create a program.
- You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook).
- **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.**

Documentation Policy:

- You must document all help received from any source other than your instructor.
- The documentation statement must explicitly describe WHAT assistance was provided, WHERE on the assignment the assistance was provided, and WHO provided the assistance.
- If no help was received on this assignment, the documentation statement must state "NONE."
- If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
- **Vague documentation statements must be corrected before the assignment will be graded and will result in a 5% deduction on the assignment.**

Turn-in Policies:

- On-time turn-in is at the specific time listed above.
- Late penalties accrue at a rate of 25% per 24-hour period past the on-time turn-in date and time. The late penalty is a cap on the maximum grade that may be awarded for late work.
- There is no early turn-in bonus or extra credit for this assignment.

0. OBJECTIVES

Upon completion of this programming exercise, students will be able to

- read data from an input file;
- write data to an output file;
- use and manipulate a list data structure;
- utilize Python as a web scripting language.

1. BACKGROUND

In this programming exercise you will write a Python program that reads data from an input file, uses that data to generate an html page to be rendered by a web browser, and writes data updated by the web page back to the data file. The html page to be created is the "Random Student Picker" application that should be familiar to you if your instructor has been using it in class. One very important aspect of this programming exercise is that you will not run your program directly from the IDE you are using to edit your code. Instead, your program will be executed by a web server and the results displayed as a web page in a web browser. Fortunately, Python makes all of this quite simple.

Internet applications have become so ubiquitous that we may not think much about all the different pieces involved. From the user's perspective, the first piece is a web browser, or client, such as Google Chrome, Mozilla Firefox, or Microsoft Internet Explorer. On the other end is a web server, of which there are implementations on many platforms using many software packages. The data transferred from the server to the client is generally HTML, or Hyper Text Markup Language. In the simplest form, the server holds static html pages that never change and are sent, as is, to the client when requested. For example, the file `hello.html` on a web server may contain the following text:

```
<h2>Hello, World!</h2>
```

When requested, this html would be sent from the server to the client and the text, "Hello, World!" would be rendered with Heading 2 style formatting and appear as follows:

Hello, World!

Static web pages such as the above are certainly useful, but modern web applications require dynamic html pages. A dynamic web page is one that is constructed by the web server when it is requested. For example, a web page may show the current date. In this case, the file `date.html` on the web server may contain something like this:

```
<h3><script>get_date()</script></h3>
```

The `<script>` tags in the html might[1] indicate to the server that it should run the `get_date()` function and send the actual date in the html to the client, which would then be rendered with Heading 3 style formatting as follows:

Thu Sep 18 2014 13:54:01 GMT-0600 (Mountain Daylight Time)

Additionally, it is possible for the client to send parameters to the server to further specify how the dynamic web page should be constructed. For example, the color of a fleece jacket may be specified in a URL as follows (the italicized and underlined text below shows the parameterization syntax):

`http://www.amazon.com/fleece_jacket.html`*`?color=green`*

The server would then use this `color` parameter, with value `green`, to determine what content to include in the html to be returned and displayed in the client's web browser.

[1] This isn't exactly how such a script would be written, but it is sufficient to illustrate the topic.

2. GETTING STARTED

Python includes a simple web server that can be started on your own machine, allowing development of web applications in Python with one machine acting as both client and server (and not even requiring an Internet connection). Once the web server is started, accessing the URL `http://localhost:8000/` will display the requested web pages, starting with `index.html`. (Note: The 8000 in the above URL is a required port number. It is somewhat arbitrary, but seems to be standard in testing situations.)

To get started, download the file `pex2.zip` from the Resources Page of our course website on Piazza and unzip the contents of this file somewhere on your machine. (Note: Be sure you right-click on the file and unzip it rather than double-clicking on the zip file and viewing the contents; ***it must be unzipped***!) Once unzipped, open the folder and double-click on the `http_server.bat` file. This should open a command window and launch Python's simple web server. Now, in your web browser, open the URL `http://localhost:8000/`. You should see a web page with a CS 210 – PEX 2 title and several links.

3. PYTHON GENERATING HTML

In your web browser, click on the first link on the index page just opened to view the page generated by the Python program in the file `picker.py`. You should see in your web browser a page with various images and links that is similar to the Student Picker your instructor may have been using this semester. Now view the source of the web page in your browser and notice it is nothing but plain HTML. This may seem strange since the `picker.py` file contains Python source code, not HTML. In fact, try running the `picker.py` program directly in your Python IDE instead of the web browser; the output to the console window is the exact same HTML you see as the source of the web page. This is because the web server executes the Python program and sends only the output of the program to the client web browser. Ta da! This is how dynamic web pages are created.

This `picker.py` file is the main file you will work with for this programming exercise, so you also need to open the file in your IDE to be able to view and edit the source code. Viewing the source code in `picker.py` in your IDE you will notice you have been given a functioning template to begin your work. Of particular interest is the `build_html_string()` function that creates and returns a string containing valid HTML. This is called in the last line of the `main()` function to output the string returned from `build_html_string()`. Again, executing this program, as is, from within your IDE, the output to the console window will be the HTML string. However, if this program is executed by a web server, the resulting HTML string is sent to the client web browser and then rendered as a web page.

Your task for this programming exercise is to modify the `build_html_string()` function such that, using the algorithm described below, it selects a student from a data file and displays that student on the web page (along with the other, updated, data shown on the template web page).

4. CGI PARAMETER PASSING

As mentioned previously, it is possible for a web client to send parameters to a web server in the URL. The Python `cgi` module includes functionality to retrieve these parameters. The file `hello.py` has been included to demonstrate how these parameters are passed in the URL and how they are obtained by the program using the `cgi` module. (CGI stands for Common Gateway Interface, a standard method used to generate dynamic web content.)

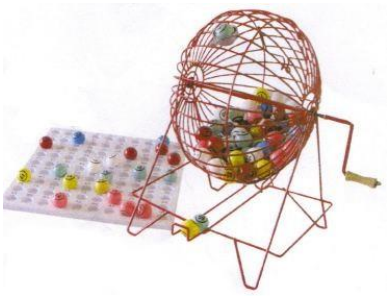
From the http://localhost:8000/ page, click on the links on the bottom and look carefully at the contents of both the web page displayed and the URL. Now open the `hello.py` file in your Python IDE and look carefully at the source code. The `cgi.FieldStorage` object provides direct access to the key/value pairs in the URL.

This `hello.py` file is provided for reference and demonstration purposes only; you do not need to modify or submit this file for this programming exercise. In fact, in the `picker.py` file you are provided, the two necessary parameters have already been extracted from the URL. The `hello.py` file also contains example Python code to generate multiple rows in an HTML table from data in the program, which may prove useful.

5. WEIGHTED RANDOM STUDENT SELECTION

The data file provided for this programming exercise contains data for several students with one line of data per student. Each line contains three pieces of information: student name, number of times the student has successfully presented, and an image file name for the student. Your program must read this data file, store the contents in an appropriate data structure, and use the data to select the next student to present. Additionally, when a student successfully presents, the contents of the data must be updated and re-written to the file.

A random, yet weighted, selection strategy will be used such that the student who has successfully presented the least number of times will have the highest probability of being selected to present. More specifically, we will use a selection strategy that somewhat follows the NBA Draft Lottery. Traditionally, professional sports leagues have determined the order of the following year's player draft based on the inverse of the final results of the current year. The goal of the NBA draft lottery is to prevent under-performing teams from intentionally losing games in order to secure the highest draft pick. That said, the worst team should still have a higher chance of obtaining the highest draft pick even though it is not guaranteed.

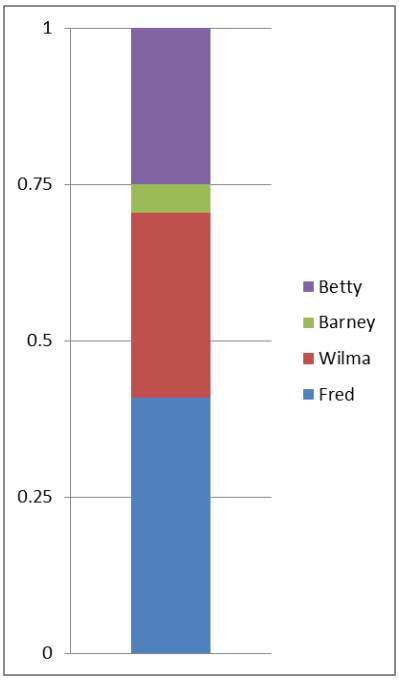


The goal of the NBA draft lottery is accomplished using Ping-Pong balls and a Bingo-style method of selecting balls from a drum. If there are 12 teams in the draft lottery, the worst team gets 12 balls, the next worst team gets 11 balls, etc., and the best of the 12 teams in the draft lottery gets one ball in the drum. Thus, the worst team has a higher probability of being awarded the first pick in the draft.

Our method will be similar with each student's probability of being selected being inversely related to the number of times the student has successfully presented. More specifically, consider four students, Fred, Wilma, Barney, and Betty, who have presented 3, 8, 19, and 10 times, respectively. Since this is a total of 40, if the distribution were uniform we would expect each student to have presented 10 times.

To get started, there must always be one Ping-Pong ball for each student in the drum. When a student successfully presents, their total is updated and the number of Ping-Pong balls in the drum is increased by one. Thus, the total number of Ping-Pong balls in the drum for each drawing will be one for each student plus one for each of the previous picks. In this example, there would be 44 balls (4+3+8+19+10) in the drum for the next pick.

Of those 44, Fred would have 18. This is calculated by starting with the expected number of times Fred would have presented in a uniform distribution (10), adding the number of times below the expected value (7), and then adding the one Ping-Pong ball that every student starts with. Similarly, of the 44 balls in the next drawing, Wilma would have 13 (10+2+1) and Betty would have 11 (10+0+1). Finally, Barney would have only 2 Ping-Pong balls in the next drawing since he has presented more than the expected number of times (10+(-9)+1).



On the Resources Page of our course website you will find a spreadsheet, PEX2_Lottery.xlsx, with two sheets showing four and eleven students. Use these spreadsheets to help understand how the probability for each student is calculated. Change the values in the Picks column and observe how the probabilities change. Make sure you understand how this works before attempting to code your solution.

Finally, once you understand how to calculate these probabilities, a randomly generated number between 0.0 and 1.0 can be used to select a student. Specifically, in the previous example, Fred, Wilma, Barney, and Betty have probabilities of being selected of 0.41, 0.30, 0.04, and 0.25. Think of these values as a stacked column chart and the random number between 0.0 and 1.0 as a dart, where is it more likely to hit?

6. FUNCTIONALITY

The basic functionality of your program is to read the data file, apply the above algorithm to select a student, and then generate the necessary HTML to show that student in a web browser as the selected student.

Additionally, your program must detect when the `prev` parameter has been included in the URL and update the data file to indicate that student has successfully presented. After doing so, a new student is selected using the above algorithm and displayed in the web page. Note: This should be the only time the data file is written!

Further, when the `next` parameter is included in the URL, your program should show that student without using the selection algorithm and no student is updated as having presented. Note: The `prev` and `next` parameters will never be passed to the program at the same time!

Finally, the HTML generated by your program must properly update all links with appropriate `prev` and `next` parameters where necessary.

7. HELPFUL HINTS

Debugging a web-based application can be a bit more challenging since the program executes on the web server and thus, `print` statements do not display output in a console window; the output of any `print` statement in your code will become part of the HTML string to be rendered by the web browser. Thus, consider the following debugging strategies:

- First, if your web page does not render properly (especially if it is completely blank), look in the command window launched by `http_server.bat`. If there was an error in your Python program, the error message should show in this window in the same format as it would in your IDE's console window, including the line number where the error was generated.

- Next, you may use `print` statements in your code, but realize the output will be sent to the web browser as part of the HTML to be rendered as a web page. Thus, surround the values you would like to print with the HTML comment delimiters as follows:

```
# Show variable stuff on web page.
print( '<!-- {0} -->'.format( stuff ) )
```

  Then use the web browser to view the source of the HTML page and you will see these values in the HTML, as comments, while the browser is still able to properly render the page.

- Last, the `picker.py` program can be executed in your IDE as a stand-alone program, as we have been doing all semester. The result of executing your program will be the HTML string printed to the console window (along with any other debugging information you choose to print). This method also allows use of the IDE's debugger.

The `picker.py` template has been designed to minimize the required knowledge of HTML. However, you may find it helpful/necessary to understand some HTML basics. To do so, run `picker.py` in your IDE and then copy/paste the results from the console window into a file named `results.html`. Look through the HTML in an editor that does HTML syntax highlighting and open the HTML in a web browser. Try modifying some of the HTML and re-loading the page in the web browser. The http://www.w3schools.com/ website is a good HTML reference.

8. REQUIREMENTS

- Read the above Help Policy carefully and ask your instructor if you have any questions.
- Include your ***detailed documentation statement*** at the top of the `picker.py` source file.
- Your program must use appropriate functions to decompose the problem.
- Your program must follow generally accepted programming practices including arranging your source file to use a `main()` function and avoiding global variables.
- Your program must be properly documented with comments including a program header comment at the top of the file and a function header comment for each function. Additionally, include comments within your code explaining individual tasks/logic. (Note: this does not mean every single line of code requires a comment!)
    - The program header comment must include your name, your section day/period, your instructor's name, a brief description of the contents of the file, and your detailed documentation statement. This comment must be written at the very beginning of the file and each line must begin with a # followed by a space.
    - Each function header comment must include a brief description of the function, the name, type, and a brief description of each parameter, a description of the value returned by the function (if any), and any necessary pre-conditions and post-conditions required by the function. These comments must be written as a multi-line string immediately following the `def` line, before any code.
    - The descriptions in each of the above comments must be written with properly punctuated complete sentences including correct spelling and grammar.
    - For an example, see the Style Sample on the Resources Page of our course website.

- Your program must follow generally accepted style guidelines to include:
    - Appropriate indentation and use of white space (space characters and blank lines) to make your code easier to read. Avoid excessively long lines that wrap around to multiple lines in the editor.
    - Use of descriptive names for variables, constants, and functions and consistent formatting of variable, constant, and function names (e.g., first letter capitalized, underscores between words, etc.)
    - For further details and style guidelines, see the following Python Style Guide at https://www.python.org/dev/peps/pep-0008.
- Use the Moodle website to submit a **single file** which is your `picker.py` file.

CS 210 – PEX 2 Grade Sheet

Name:_____

Section:_____

Criteria		Points Earned	Points Available
Documentation (18 points)			
Standard program header comment block			**6**
Standard function header comment blocks			**6**
Appropriate commenting throughout			**6**
Readability and Maintainability (42 points)			
Program formatting			**6**
Meaningful / consistent identifier names			**6**
Appropriate use of named constants			**6**
Appropriate functional decomposition			**12**
Adheres to generally accepted programming practices including arranging the source file to use a `main()` function, avoids use of global variables, etc.			**12**
Functionality (50 points)			
Reads data file and stores data in an appropriate data structure			**10**
Properly applies selection algorithm to select the next student			**10**
Updates data file when (and only when) **`prev`** parameter is specified			**10**
Loads a specific student when **`next`** parameter is specified			**10**
Creates correct HTML with all required data, links, and parameters			**10**
	Subtotal:		**110**
Adjustments	**Vague/Missing Documentation:**		**– 6**
	Submission Requirements Not Followed:		**– 12**
	Late Penalties:		**25/50/75%**
	Total w/adjustments:		

Comments: