EECS 1015: Final programming exam (Lists, tuples, dictionaries, sets, assertions, classes, multi-file)
Assigned: Dec 10, 2020 (Thursday)
Due date: Dec 15, 2020 (Tuesday) [11.59 pm Eastern Time – No extensions/No late submissions!]

#Important for the final programming exam
1) You must submit your final exam via web-submit to the "final" folder (see instructions on the last pages).

2) Please make sure you correctly submit your files ***(two files*** – `final.py, ultilities.py`).

3) Read the task descriptions carefully to understand everything you need to do. There are five (5) tasks.

2. INSTRUCTIONS

1) You have 5 days to complete this take-home final exam. There are no extensions or late submissions.

2) The final programming questions incorporate components from topic 2 to topic 10.

3) The final has five (5) tasks. Please complete each task.

4) This final is graded out of 100 points.

5) IMPORTANT: Your program must use the starting code provided below (see trinket.io link)

6) Please name your submission: final.py, utilities.py

STARTING – two files provided with starting code.

(1) `final.py` – main() program and the five tasks.

(2) `utilities.py` – variables needed for the tasks.
You should also write your class in the utilities.py file (see Task 4)

You can cut-and-paste from here: https://trinket.io/python/0261336836
(On the trinket site there is a tab to access the two files – please remain main.py as final.py)

A video describing the final is available here: http://www.eecs.yorku.ca/~mbrown/EECS1015final.mp4

3. TASKS 0-4

TASK 0 (0 points correct, -10 points deducted if you forget or have the wrong information)

The function `task0()` should print out the following information:

```
Final Exam - EECS1015
Name: Your Name
Student ID: Your York ID
email: youremail@aol.com
```

See the next pages for each tasks 1 to 4's description.

TASK 1 - Word frequency in a song lyric [15 pnts]
Topics covered: variables, expressions, control structures, string processing, functions, and lists.

In the `utilities.py` file, there is a variable named `lyrics` that is bound to a long string that has some of the words to the song "Happy." Note that the string includes newline characters, commas, and periods.

In the `main()` function, you should pass the `lyrics` variable as an argument to the function `task1()`. Your task is to extract the words from the `lyrics` string and count the number of times each word appears in the string. Follow the procedure below. You are welcome to introduce additional functions. If you write additional functions, write them in the `final.py` file (not in `utilities.py`).

Perform the following steps:

String pre-processing
(1) Convert the string passed to your `task1()` function to lower case
(2) Remove all commas and periods from the string.
Counting the words and printing out the information
(3) Split the string to find all the words.
(4) For each word, determine the number of times it appears.
(5) Print out each word, its count, and a string of * representing the count.
For example, if the string `"the"` appears three (3) times and the string `"you"` appears four (4) times, print:

```
the [3] ***
you [4] ****
```

Hint: When printing out the word from the lyrics, you can use "%10s" in the string formatting. This notation will result in the string always taking ten (10) characters.

Here is a *partial* output from Task 1—see also the accompanying video.

```
--- Task 1 ---
      what  [3] ***
     along  [4] ****
       i'm  [6] ******
     she's  [1] *
       the  [3] ***
      take  [1] *
      roof  [1] *
           ....
       you  [7] *******
```

Note: There are many ways to perform this task. You may find using the "set" data type discussed in lecture 10 useful. Set can be used to find the unique items in a list. For example:
`x = set([10,20,10,20,30]) ->  x = {10,20,30}`.
Also, remember that you can always convert a set to a list as follows:
`x = list({10,20,30}) -> x = [10,20,30]`

Task 2 – "Invert" a dictionary's keys and values [25 pnts]
Topics covered: variables, expressions, control structures, functions, lists and dictionaries.

In the `utilities.py` file, there is a variable named `nameDict` that is bound to a dictionary object. The dictionary object's keys are strings (representing a person's name), and the value associated with each key is the person's age (e.g., `"Abdullah":18, "Su":21, …, "Priya":18`).

In the `main()` function, you should pass the `nameDict` variable as an argument to the function `Task2()`. Task2 function should create a new dictionary that "inverts" the keys and values in `nameDict`. In particular, your new dictionary will have keys that are the ages; the associated value for each key will be a list of names.

Once you have "inverted" the dictionary, you should do the following.

(1) Print out the new dictionary variable (this way, we can see you constructed it correctly)

(2) For each key in the new dictionary, print out the age and then each person's name in the list as shown below.

You are welcome to introduce additional functions. If you write additional functions, write them in the `final.py` file (not in `utilities.py`).

Here is a *partial* output from Task 2—see also the accompanying video.

```
{17: ['Dirk', 'Sarah', 'Bosko', 'Cameron'], 18: ['Mahesh', 'Abdullah', 'Franck',
'Xavier', 'Maxim', 'Priya'], 19: ['Kai', 'Bailey', 'Ollie', 'Parsa', 'Ming',
'Javier'], 20: ['Jol', 'Seonjoo', 'Beatrice'], 21: ['Su', 'Abbo', 'Olivia'], 22:
['Pravel', 'Urzula'], 23: ['Katja'], 24: ['Svetlana']}
Age 17
  Dirk
  Sarah
  Bosko
  Cameron
Age 18
  Mahesh
  Abdullah
  Franck
  Xavier
  Maxim
  Priya
  …
Age 24
  Svetlana
```

Task 3 – Poor person's graphics --- draw a circle in a 20x20 raster (2D lists) [30 points]
Topics covered: variables, expressions, control structures, functions, and lists of lists, testing (assert)

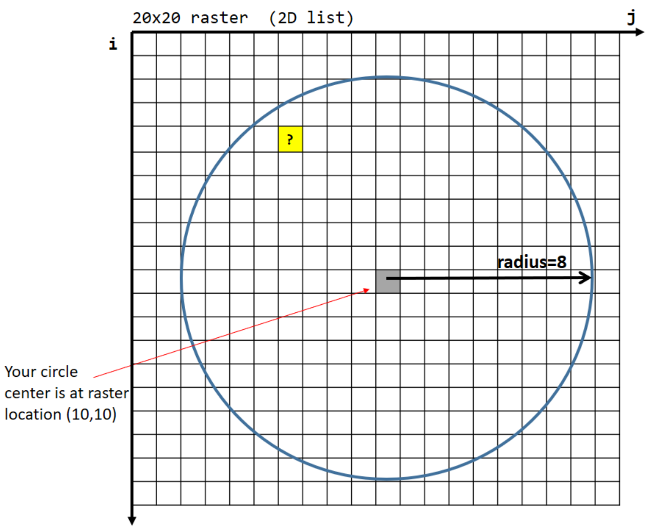
In the `utilities.py` file, there is a variable named `raster` that is bound to a list of lists. The raster variable has 20 lists, each list with 20 empty strings. This "list of lists" is effectively a 2D list that logically makes up a 20x20 grid. In computer graphics, such grids are called a raster. Our simple drawing program will place either a `"*"` or a `" "` (space) in each location of the 20x20 grid. This can be done by access the element as follows: `raster[5][10] = "*"` would place a `"*"` at the 6th row and 11th column of the raster; `raster[3][3] = " "` would place a `" "` at the 4th row and 4th column of the raster. Task 3 goal is to draw a circle in the 20x20 raster based on a user-supplied circle radius between 1 and 9.

In the `main()` function, you should pass the `raster` variable as an argument to the function `Task3()`.
Your function should work as follows:

(1) Ask the user for a radius from [1 to 9]. If the user inputs a value that is not 1-9, then raise an **assertion error** with the following statement: `"Radius must be between 1-9"`.

(2) Loop through the 20x20 raster to draw the circle using the algorithm sketched out below:

```
loop through i -> 0 to 19
loop through j -> 0 to 19
 at raster location i,j
  # convert i,j to an x,y coordinate by
  # subtracting 10 from the i,j
  #  (subtracting 10, centers the circle at 10,10)
  x = i - 10
  y = j - 10
  if sqrt($x^2$ + $y^2$) <= radius
     place a '*' in the i,j location on the raster
  otherwise
     place a ' ' in the i,j location on the raster
```



The sqrt() function from the math module has been imported for you in the final.py file.

For example:
Consider the yellow raster location (i=4 and j=6) with in the figure above. The radius of the circle to draw is 8.

```
Let's determine if the raster location i=4 and j=6 should be a " " or a "*"?
Follow the formula above:
x = 4-10 = -6
y = 6-10 = -4
sqrt(-6*-6 + -4*-4) = 7.22
7.22 is less than or equal to 8, so raster[4][6]="*"
```

(3) After you draw the circle, ask the user if they would like to do it again?

See sample output on the next page.

Example output for Task 3 (also see accompanying video)

```
--- Task 3 ---
Radius [1 to 9]: 5





          *
       *******
      *********
      *********
      *********
     ***********
      *********
      *********
      *********
       *******
          *




Try again: Y/N? y
Radius [1 to 9]: 9

          *
      *********
     ***********
    *************
   ***************
  *****************
  *****************
  *****************
  *****************
 *******************
  *****************
  *****************
  *****************
  *****************
   ***************
    *************
     ***********
      *********
          *
Try again: Y/N?
```

Task 4 – Robust statistics for a corrupted list (Class/Objects) [30 pnts]

Topics covered: variables, expressions, control structures, functions, lists, classes/objects

In the `utilities.py` file, there is a variable named `numList` that is bound to a list of numbers. For Task4, you will write code for a class named `listStats` that is used to compute the mean (also called the average), trimmed mean, and median of a list. The trimmed mean and median are known as "robust statistics" because they can give good answers even when some of the data in the list has been "corrupted". Task4 will examine how the mean, trimmed mean, and median perform on corrupted input.

Your `listStat` class should have the following methods that provide the described functionality:

`__init__( parameters: numList, corruption_level )`

This method is the object's constructor or initializer. The constructor is passed a numList and a "corruption level". The corruption level is a value between 0 and 100.
Create an instance variable that stores a *copy* of the numList.

Next "corrupt" the numList as follows:

```
For each item in the numList, generate a random number between 1 and 100.
  If the random number is <= the corruption_level parameter, then corrupt the item as follows:
    Generate another random number between 1 and 100.
    If the number is > 50, set the item in the list to -1; otherwise set the item to 100.
```

`computeMean()`

This method computes the mean (i.e., standard average) of the corrupted list.
The mean is the sum of all items in the list divided by the number of items.

Print out the result as `"Standard Mean   XXX"` where `XXX` is formatted to have 2 decimal places.

`computeTrimmedMean()`

A "trimmed mean" is computed as follows.

(1) sort the list
(2) Remove the first 10 items and last 10 items from the list
    The removal of these items is where the term "trim" comes from, which means to "cut" or "remove."
(3) Now, compute the average of the *new* list.
    - Note: sum only the items in the trimmed list and divide by the number of items in the trimmed list.

Print out the result as `"Trimmed Mean     XXX"` where `XXX` is formatted to have 2 decimal places.

`computeMedian()`

The median is computed by sorting the list and taking the middle value.
The middle value is the length of the list divided by 2 using integer division (//).
Take care to make sure this method sorts the full corrupted list and not the trimmed list.

Print out the result as `"Median          XXX".` Note that the median will be an integer.

`printStatistics()`

This method prints out the corruption level and the corrupted list (see output example on next page).
This method then calls the computeMean(), computeTrimmedMean(), and computeMedian() methods.
Recall that you call a method within another method using the self keyword (see Lecture 9 slides 46-47 for an example in the "putting it all together" example). The mean/trimmed mean/median methods will print out their results as described above.

Class code and task4() code

<u>IMPORTANT: write your class in the `utilities.py` file.</u> In `task4()` in the main file (`final.py`), declare three `listStats` objects with the following corruption levels (0, 15, and 30). For each of these three objects, call the `printStatistics()` method as follows:

```
obj1 = listStats(numList, 0)  # no corruption
obj2 = listStats(numList, 15) # 15% chance of corruption
obj3 = listStats(numList,30)  # 30% chance of corruption
obj1.printStatistics()        # print stats
obj2.printStatistics()        # print stats
obj3.printStatistics()        # print stats
```

Example output for Task 4 (also see accompanying video)

```
--- Task 4 ---

Corruption level 0%
[21, 20, 29, 16, 16, 19, 26, 19, 26, 23, 19, 25, 16, 18, 18, 18, 21, 15, 18, 14, 22, 17,
32, 30, 22, 28, 26, 17, 19, 16, 17, 17, 19, 21, 20, 17, 18, 34, 23, 18, 23, 22, 17, 22,
16, 11, 21, 26, 20, 27]
Standard Mean  20.70
Trimmed Mean   19.90
Median         20
Corruption level 15%
[21, 20, 29, 100, 16, 19, 26, 19, 26, 23, -1, 25, 16, 18, 18, 18, 100, 15, 18, 14, 22,
17, 32, 30, 22, 28, 100, -1, 19, 16, 17, -1, 19, 21, 20, 17, 18, 34, 23, 18, 23, -1, 100,
-1, -1, 11, -1, 26, 20, 27]
Standard Mean  24.28
Trimmed Mean   20.03
Median         19
Corruption level 30%
[100, 20, 29, 16, -1, 19, 26, 19, 100, 23, 19, 100, 16, 18, 18, 100, 21, 15, 18, 14, 22,
100, 32, 100, 22, -1, 26, 17, -1, 16, 17, 17, -1, 21, 100, -1, 18, 34, 23, 18, 23, 100,
17, 22, 16, 11, 21, -1, 20, 27]
Standard Mean  30.50
Trimmed Mean   20.30
Median         20
```

Note how items are corrupted as either -1 or 100.

Notice how the trimmed mean and median give similar results even when the list is corrupted.

FINAL COMMENTS: For tasks 1-4 you may introduce additional functions (or, in the case of task4, additional methods); however, it is not required and will not affect your grade either way.

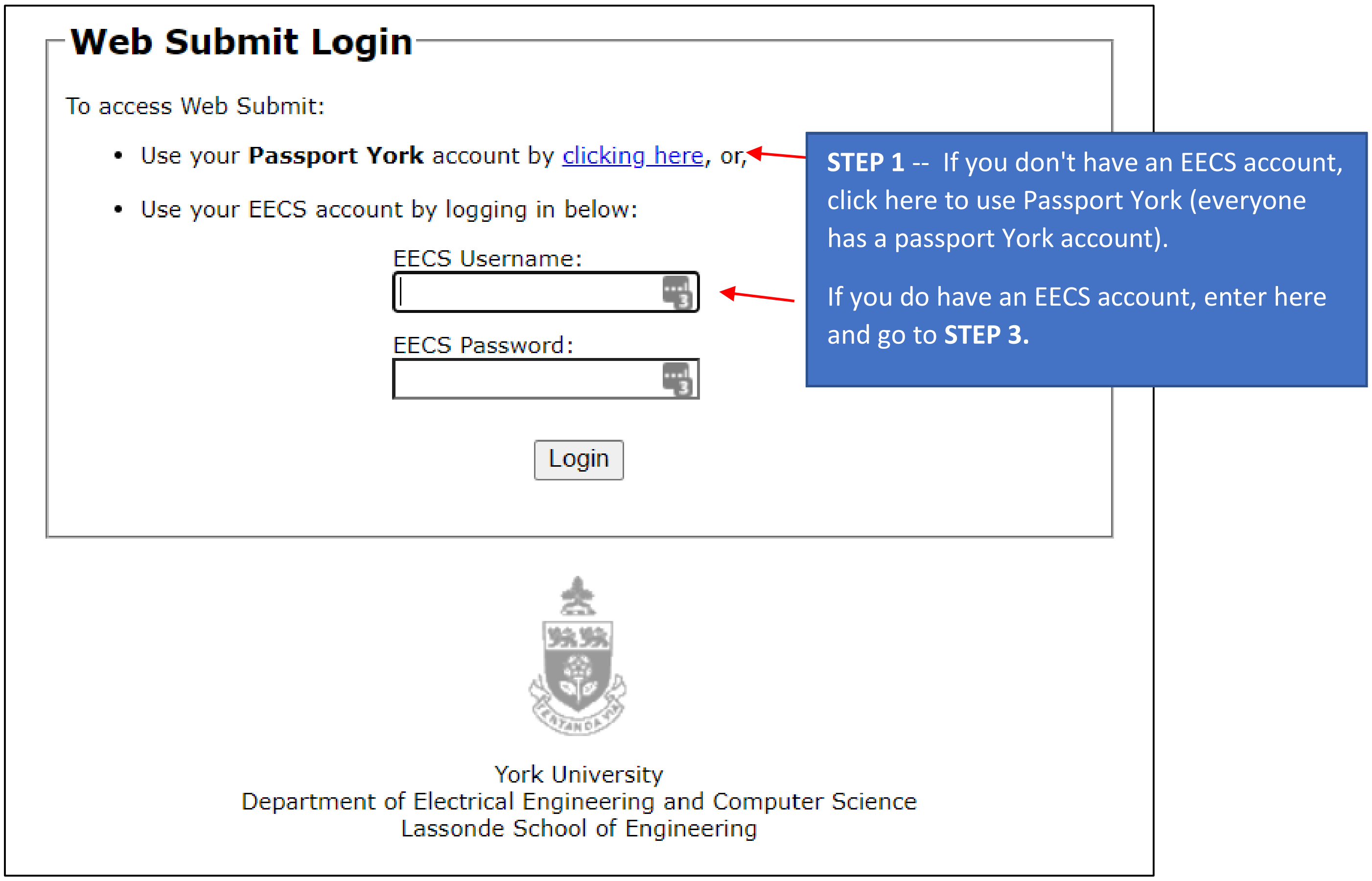
See pages below on how to submit your final exam's code.

MAKE SURE TO SELECT **final** with websubmit.

4. SUBMISSIONS (EECS web-submit)

You will submit your final using the EECS web submit.

Click on the following URL: https://webapp.eecs.yorku.ca/submit



Web Submit Login

To access Web Submit:

- Use your **Passport York** account by clicking here, or,
- Use your EECS account by logging in below:

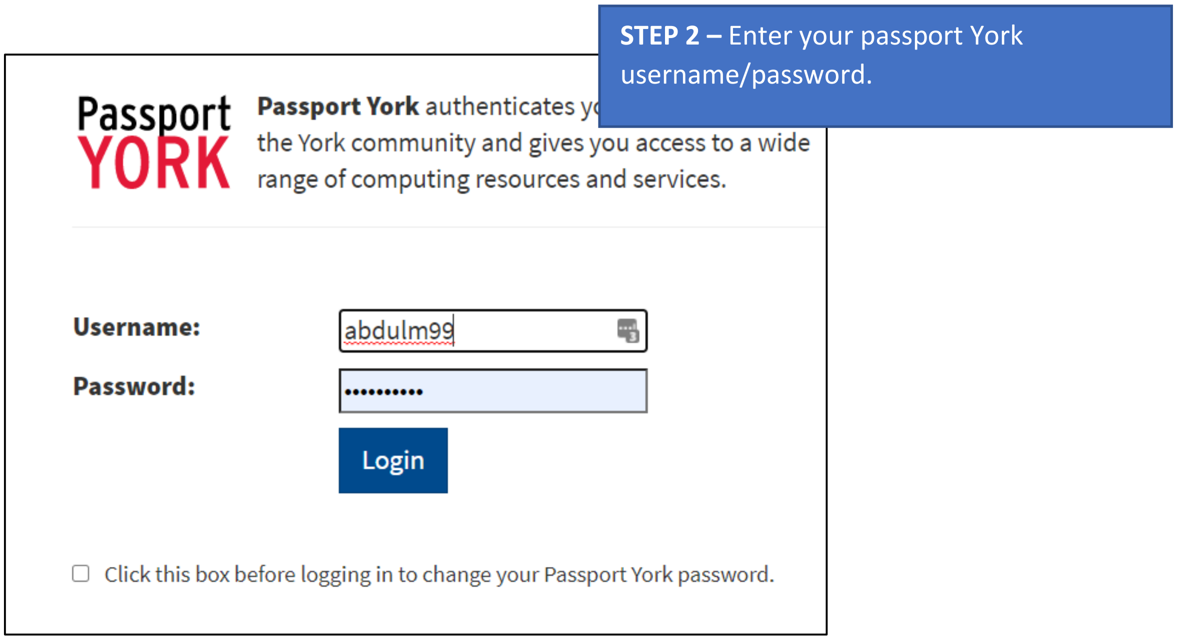
EECS Username:

EECS Password:

York University
Department of Electrical Engineering and Computer Science
Lassonde School of Engineering

STEP 1 -- If you don't have an EECS account, click here to use Passport York (everyone has a passport York account).

If you do have an EECS account, enter here and go to **STEP 3.**



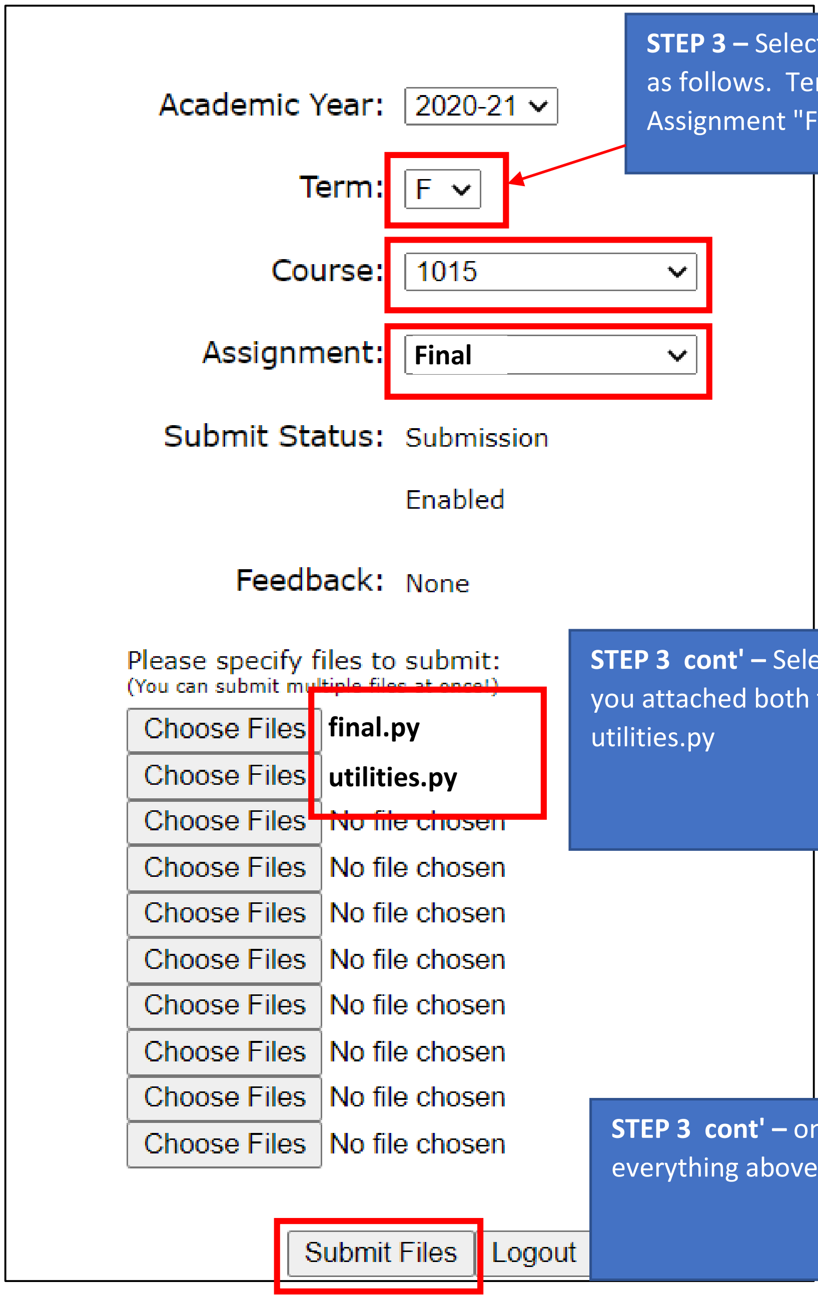
STEP 2 – Enter your passport York username/password.

Passport York authenticates y the York community and gives you access to a wide range of computing resources and services.

Username: abdulm99

Password: ••••••••••

☐ Click this box before logging in to change your Passport York password.



Academic Year:	2020-21
Term:	F
Course:	1015
Assignment:	**Final**
Submit Status:	Submission Enabled
Feedback:	None

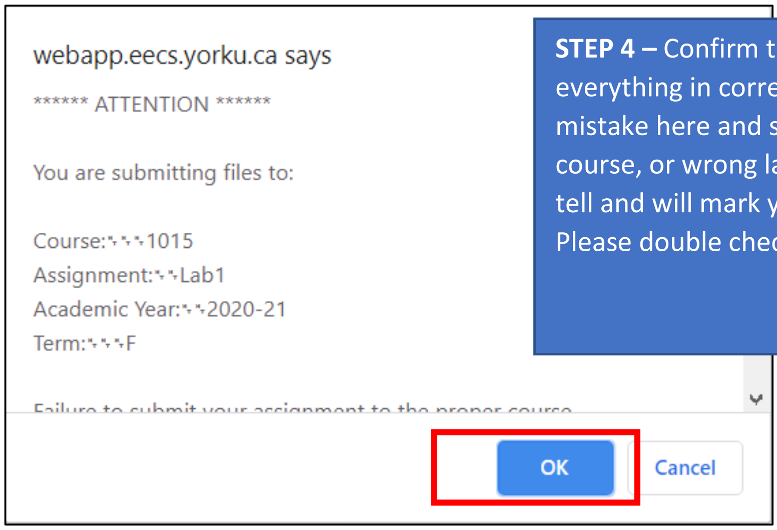
Please specify files to submit:
(You can submit multiple files at once!)

Choose Files	**final.py**
Choose Files	**utilities.py**
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen

STEP 3 – Select the correct menu option as follows. Term "F", Course "1015", Assignment "Final".

STEP 3 cont' – Select your file. Make sure you attached both files final.py and utilities.py

STEP 3 cont' – once you have entered everything above, click "Submit Files".



webapp.eecs.yorku.ca says

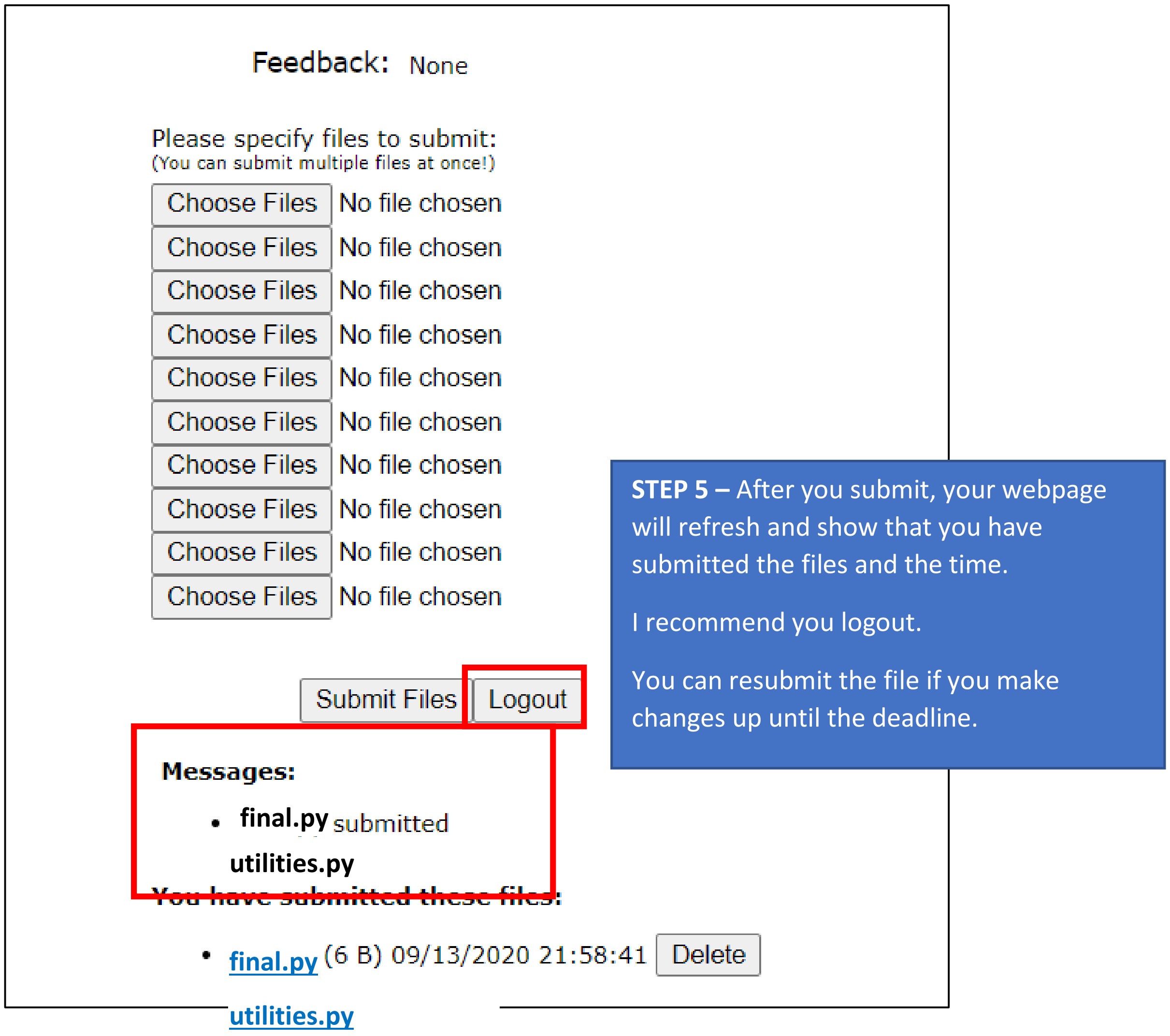
****** ATTENTION ******

You are submitting files to:

Course: 1015
Assignment: Lab1
Academic Year: 2020-21
Term: F

Failure to submit your assignment to the proper course

STEP 4 – Confirm that you have entered everything in correctly. If you make a mistake here and submit to the wrong course, or wrong lab, we won't be able to tell and will mark your lab as not submitted. Please double check before clicking OK.



Feedback: None

Please specify files to submit:
(You can submit multiple files at once!)

Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen
Choose Files	No file chosen

Messages:

- **final.py** submitted

utilities.py

You have submitted these files:

- final.py (6 B) 09/13/2020 21:58:41 Delete

utilities.py

STEP 5 – After you submit, your webpage will refresh and show that you have submitted the files and the time.

I recommend you logout.

You can resubmit the file if you make changes up until the deadline.

For more details on websubmit, see EECS department instructions:

https://wiki.eecs.yorku.ca/dept/tdb/services:submit:websubmit