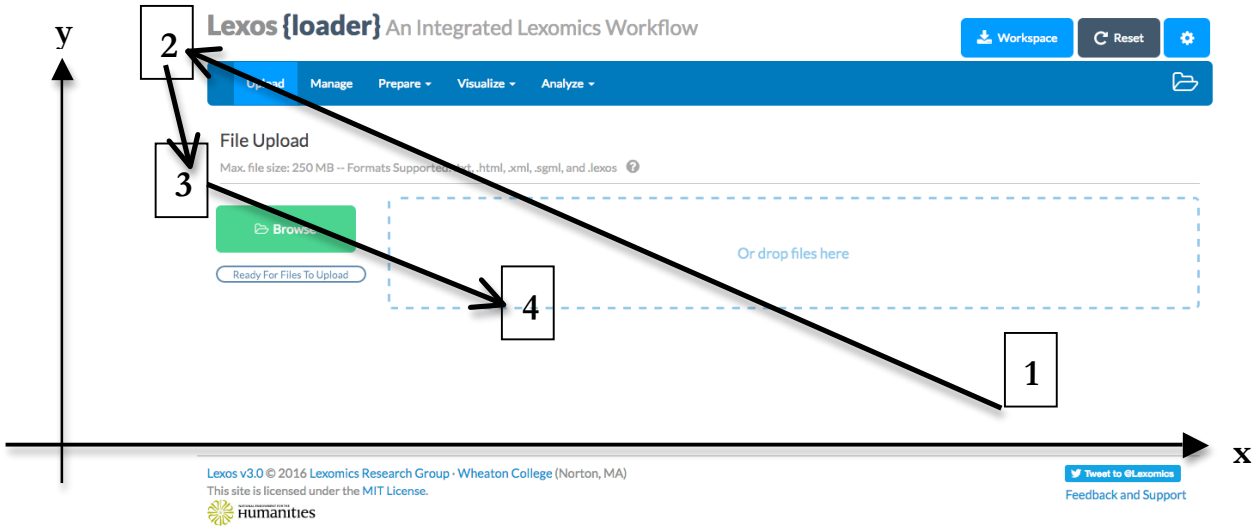
Tracking your Gaze
Programming with eye-tracking data
Due Monday, October 3

Summary
Our initial program worked with a limited set of data from a pair of Tobii Pro eye-tracking glasses, think EyeTracking *v1.0*. These glasses combine cool with (potential) creep as they (i) take video of wherever you are looking and (ii) save the (x,y) coordinates of *exactly* where you are looking. A number of research areas are applying these glasses, for example: (a) user experience and interaction (tracking where you are looking as you play a video game or use a new app); (b) marketing and consumer research (tracking where a consumer is looking on a shelf while shopping or where they are looking at an advertisement); (c) infant and child research (tracking where early readers are focusing on the page); and (d) psychological and neuroscience (tracking how and why eye movements are made).

This programming assignment will enhance your initial program: EyeTracking *v2.0*. In this version, your program will be able to read in an entire file of (x, y) coordinates (recall that initially we could only four (x,y) pairs). The figure below shows the opening screen to our Lexomics Research Group's recent *v*3.0 release of our *Lexos* software. The numbers and (x, y) coordinates indicate the first four locations on the path where I gazed as recorded by the eye-tracking glasses while viewing the browser window at: http://lexos.wheatoncollege.edu.



The eye-tracking glasses export the (x, y) gaze coordinates to a comma-separated-value (.csv) file (e.g., "`eyetracking1.csv`") that can be opened in Excel *and/or* your Python program. The first four (x, y) coordinates (of potentially hundreds, if not thousands of points, are shown here. Using these (x,y) coordinates, your task is to write a Python to compute and output metrics about all the gaze points in a file.

`eyetracking1.csv`

	A	B
1	Gaze point X	Gaze point Y
2	90	10
3	5	40
4	10	30
5	50	25

INPUT
The input to this program comes from two sources: (i) standard input (`stdin`), that is the **keyboard** and (ii) a **file** of pairs of (x,y) coordinates, each line in the file containing two Real (floating point) values per line. The initial line of the file contains header labels on the columns.

Notes:
(1) I have given you a set of input files so you can test your program on multiple data sets. Imagine in the future that you could save your own data with the eye-tracking equipment and then use that file with your

Python program! Note that you can also create a totally new file (in Excel) and create your own sample input file to test (but take care to save your file as comma-separated-value (.csv) file when saving).

(2) You should test your program with multiple files to make sure your program works no matter the (x, y) coordinates. What possibly could go wrong? If you discover a potential problem, you must trap those conditions and print appropriate messages.

(3) When you submit your program, you must submit with the filename "`eyetracking1.csv`".

ANALYSIS

Your program must compute and print the **distance**, **direction of gaze** (e.g., "look down and to the right", "look horizontally to the left", "look vertically up", etc.), and **the slope** between pairs of all the gaze points (e.g. 1-to-2, 2-to-3, 3-to-4, *etc*). You should also compute the **total** and **average** distance between the fixations.

OUTPUT

The output from this program must be neat and include the following: a title, a neat summary of each of the (x,y) points in (x, y) fashion, and good messages that indicate (i) the **units** of your values and **filename** being used, (ii) **distance** between pairs of the gaze points, (iii) the **slope** between pairs of the gaze points, (iv) the **direction of gaze** (e.g., from point 2 to point 3 the person looked "down and to the right"), and after all points have been processed, (v) the **total gaze distance** and (vi) the **average gaze distance** if appropriate. You should print the infinity symbol (∞) for vertical slopes. A partial sample output is shown below:

```
==========================================
        Summary of Gaze Fixations
         (all units in pixels)
==========================================

Using data file:  eyetracking1.csv

----------------------------------
Gaze Fixation (1): ( 90.0, 10.0)
Gaze Fixation (2): (  5.0, 40.0)
Distance from (1) to (2) is: 90.14
    Up to left, slope = -0.35

----------------------------------
Gaze Fixation (2): (  5.0, 40.0)
Gaze Fixation (3): ( 10.0, 30.0)
Distance from (2) to (3) is: 11.18
    Down to right, slope = -2.00
          :
          :
Total Gaze Distance from (1) to (4) is:      141.63
Average Gaze Distance between fixations is: 4.72e+01
```

Further Directions

• You must write and call (at least) the following functions:

`distance( x1, y1, x2, y2 ):` This function accepts four, floating-point values of the (x,y) coordinates of two points. The function should return the distance between these points.

`slope( x1, y1, x2, y2 ):` This function accepts four, floating-point values of the (x,y) coordinates of two points. The function should return the slope of the line between these points. Note: this function *should NOT be called if the slope computation would cause division by zero; that is, you must check (trap) in the main() function BEFORE the call to make sure you only call* `slope()` *if it is safe to compute the slope.* Said differently, this function should not contain an IF-ELSE statement to catch division by zero since this function's "pre-condition" is that the user will *not* call this slope function with two vertically-aligned points.

`printTitle():` This function accepts no arguments and returns no values. It prints a nice title.

`printSlope(x1,y1,x2,y2):` This function accepts four, floating-point values of the (x,y) coordinates of two points. The function does not return any values. The function should print an appropriate message that indicates the direction of the person's gaze from the initial point (x1,y1) to the second point (x2,y2). You must handle all possible directions, e.g., "look down and to the right", "look horizontally to the left", "look vertically up", etc. This function should trap any potentially bad situations. This function should call the slope() function to compute the actual slope value, if safe to do so. Vertical slopes should print the infinity symbol (∞) as the slope's value.

• You must exhaustively test your program using many different files. (See the INPUT section above).

• You must use formatted output to ensure you can see levels of precision. Print (x,y) floating point (**`f`**) coordinates with one place after the decimal point (**`%5.1f`**), two places after the decimal for the distances and slopes, and use scientific notation (**`e`**) for the average gaze distance.

```
print ("Gaze Fixation (1): (%5.1f,%5.1f)" % (x1,y1) )
print ("Average Gaze Distance between fixations is: %7.2e" % avgDistance )
```

• Your program must have initial comments, giving your name, the **PURPOSE** or summary of the program (that is, what does it do or why would someone run this app), a description of the **INPUT** the program uses (from *where* and the *data types*) and a description of the program's **OUTPUT** a user should expect if they run your code**.** I have given you a template to get you started.

• Each function should contain a line of documentation that succinctly states the purpose of the function. Use the standard one-line Docstring as described at: https://www.python.org/dev/peps/pep-0257/

• Name your Python source file with *your Lastname and the assignment#*, e.g., I would name my source file: **LeBlancA3.py**

GRADING

You will be graded by the following general rubric:

Average: You finish the program and your program produces the **correct Output** for slope, distance, and direction of gaze.

Above Average: You have correct output and you meet all the Requirements and Program Details.

Superior Effort: Compute and print two other statistical values of your overall data: (i) the **minimum** and (ii) the **maximum** gaze distance.

SUBMISSION

• You only need to submit your Python source code: e.g.,`LeBlanc_a3.py` Note: you do *not* have to submit any input files since I already have them.

• Submit your Python source via onCourse by: Monday, Oct 3rd (Tuesday morning, by 4am)