CS 116 Fall 2017 Assignment 08

Due: Wednesday Nov. 22nd, 2017 at 10 am

Assignment Guidelines

- This assignment covers material in Module 09.
- For this assignment. and for each function, you need to write only contract, effects, and tests! ☺
- Do not use recursion. You may use loops and abstract list functions (`map` and `filter`)
- Submission details:
    - Solutions to these questions must be placed in files `a08q1.py`, `a08q2.py`, and `a08q3.py`.
    - You must be using Python 3 or higher.
    - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
    - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
    - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
    - For full style marks, your program must follow the Python section of the CS116 Style Guide.
    - Be sure to review the Academic Integrity policy on the Assignments page
- Download the testing module from the course web page. Include `import check` in each solution file.
    - For `check.within` use tolerance of 0.0001
- Restrictions:
    - Do not import any modules other than `math` and `check`.
    - Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza.
    - You are always allowed to define your own helper functions, as long as they meet assignment restrictions.
    - While you may use global constants in your solutions, do not use global variables for anything other than testing.
    - Read each question carefully for additional restrictions or tips.

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

1.

Write a Python function `freq_table` that consumes a non-empty list of integers (`nlst`), and returns a dictionary representing the numbers in `nlst` and their frequencies. The function should also print a basic average statement (2 lines) in the format below as presented in the examples.

For example:

`freq_table([1])` should return `{1:1}` and print

```
Average = (1*1)/(1)
Average = 1.0
```

`freq_table([1,2,3,4,5,6,4,87,98,88,98,98])`

should return

```
{1: 1, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 87: 1, 98: 3, 88: 1}
```

and print

```
Average = (1*1+2*1+3*1+4*2+5*1+6*1+87*1+98*3+88*1)/(1+1+1+2+1+1+1+3+1)
Average = 41.166666666666664
```

Notes:

- The order of the dictionary and the printed statement does not matter.
- There are in total only two spaces in each line in the printed message; before and after =

2.

All parts of this question should be submitted in one file

A term stores the courses offered in a dictionary of courses codes with their students who are already enrolled (as a list of students' ids). Examples of terms are:

```
term1={"Eng100":[1,3,4,5,7,8,9,10,11], "CS116" :[1,3,4,5,6,7,8,10],
       "Math135":[11,3,4,5,7,8,9,10,1]}
term2={"CS116":[11,12,13,14,15], "Math135":[11,12,34,56,44],
       "Drama":[11,12,34,56,44,111]}
```

a) Write a Python function `most_popular_courses` that consumes a dictionary, `term`, `(A dictionary where the key is of type Str and the value is of type non-empty list of positive Natural numbers)`, and returns a list of the most popular courses. The most popular courses are the courses with the maximal number of enrollment. For example:

```
most_popular_courses(term1) => ['Eng100', 'Math135']
most_popular_courses(term2) => ['Drama']
```

b) Write a Python function `common_courses(t1, t2)` that consumes two terms and returns a list of all courses that are offered in both consumed terms. Note, the function should not mutate neither t1 nor t2. For example:
`common_courses(term1,term2) => ['CS116', 'Math135']`

c) Write a Python function `offered_once(t1, t2)` that consumes two terms and returns a list of courses that offered in only one of the two consumed terms. Note, the function should not mutate neither t1 nor t2. For example:
`offered_once(term1, term2) => ['Eng100', 'Drama']`

d) Write a Python function `enroll_student(term, course, studid)` that consumes a term, a course code and a student id. The function prints a message as follows in the examples if the student is already enrolled or the course is not offered and returns `False`, otherwise the function mutates the consumed `term` by adding `studid` in `course`, and returns `None.` For example:
`enroll_student(term1, "CS135", 300) => False` and will print:
`The course CS135 is not offered in the provided term.`
`enroll_student(term1, "CS116", 5) => False` and will print:
`The student 5 is already enrolled in course CS116.`
`enroll_student(term1, "Math135", 300) => None` and
`term1` is mutated to:
`{'Eng100': [1, 3, 4, 5, 7, 8, 9, 10, 11], 'CS116': [1, 3, 4, 5, 6, 7, 8, 10], 'Math135': [1, 3, 4, 5, 7, 8, 9, 10, 11, 300]}`

e) Write a Python function `drop_student(term, course, studid)` that consumes a term, a course code and a student id. The function prints a message as follows in the examples if the student is not enrolled or the course is not offered and returns `False`, otherwise the function mutates the consumed `term` by removing `studid` in `course`, and returns `None`. For example:
`drop_student(term1, "Eng100", 244)=> False` and will print:
`The student 244 is not enrolled in course Eng100.`
`drop_student(term1, "Eng200", 4) => False` and will print:
`The course Eng200 is not offered in the provided term.`
`drop_student(term1, "Eng100", 4) => None` and `term1` is mutated to:
`{'Eng100': [1, 3, 5, 7, 8, 9, 10, 11], 'CS116': [1, 3, 4, 5, 6, 7, 8, 10], 'Math135': [1, 3, 4, 5, 7, 8, 9, 10, 11]}`

3.

All parts of this question should be submitted in one file

Recall from high school geometry that two points, (x1,y1) and (x2,y2),define a line in a plane. One standard equation for a line is y = mx+b where m is the slope, defined by m = (y1−y2)/(x1−x2) and b is the y-intercept (i.e., the point where the line crosses the y-axis). This form of the equation of a line does not work well for vertical lines, however (since the slope is undefined, and there is no unique y-intercept). If the line is vertical, there is a unique x-intercept which does define the vertical line. This question will be using the following class definition. **Do not copy and paste from this pdf file** (check the provided interface).

```
class Line:
    '''Fields: slope(anyof Int Float "undefined"),
               intercept (anyof Int Float)
          '''

    def __init__(self,slope,intercept):
        self.slope = slope
        self.intercept = intercept


    def __repr__(self):
        s1 = "Y = {0:.2f}X + {1:.2f}"
        s2 = "X = {0:.2f}"
        s3 = "Y = {0:.2f}"
        if self.slope=="undefined":
            return s2.format(self.intercept)
        elif self.slope==0:
            return s3.format(self.intercept)
        else:
            return s1.format(self.slope, self.intercept)

    def __eq__(self, other):
       return type(other) == type(self) and \
               self.slope == other.slope and \
               self.intercept == other.intercept
```

Complete the following Python `Line` class methods, with a Complete design recipe for each method, (all tests should be written after the end of class Line definition):

(a) `points_to_line` which consumes two distinct points through 4 parameters (`x1, y1, x2, y2`) and returns a `Line` that goes through these two points. Specifically, you should set the slope with the computed slope between the two points, and the intercept to be the y-intercept. Should your `Line` be vertical, you should set the slope of the `Line` to be "`undefined`" and set the intercept to be the intercept on the x-axis.
For example:

```
check.expect("Q3T1", Line.points_to_line(-7,6,9,6),  Line(0,6))
check.within("Q3T2", Line.points_to_line(-7,6.1,9.45,6).slope,\
             -0.006079, 0.0001)
check.within("Q3T2", Line.points_to_line(-7,6.1,9.45,6).intercept,\
              6.0574468, 0.0001)
```

(b) `perpendicular_line` which consumes `x` and `y` representing a point. Your function should return a `Line` which goes through the given point(`x,y`) and is perpendicular to the object(`self`). Again, the object (`self`) or the returned `Line` may be vertical, and should be treated the same way as in part (a). As a reminder, a perpendicular line has slope which is the negative-reciprocal to the original line (i.e., the negative-reciprocal of 2 is −1/2).
For example,
Suppose

```
L3= Line("undefined", 0)
```

Then `L3.perpendicular_line(3,4)` returns `Line(0,4)`

```
     L3.perpendicular_line(0,0) =>  Line("undefined", 0)
```

(c) `parallel` which consumes a `Line` (that is different from the object `self`). Your function should return `True` if the consumed `Line` and the object (`self`) are parallel, otherwise the function produces `False`. As a reminder, parallel lines are lines in a plane which do not meet; that is, two lines in a plane that do not intersect or touch each other at any point are said to be parallel.
Suppose

```
L1=Line(10,4)
L2=Line(10, -5)
```

Then `L1.parallel(L2)` returns `True`

(d) `intersect` which consumes a `Line` (that is different from the object `self`). Your function returns a list of length two that represents a point ([x_coordinate, y_coordinate]) that represents the intersection point between the consumed `Line` and the object `self`.
If both consumed `Lines` are parallel, then the function returns `False`.
Suppose

```
L5=Line("undefined",10)
L6=Line(0,5)
```

Then `L5.intersect(L6)` returns `[10, 5]`