

## Instructions

This assignment is to be done **INDIVIDUALLY**.

The goal of this assignment is to learn and practice the concepts covered in Sets, Dictionaries and Objects.

Any files with syntax errors will automatically be excluded from grading. Be sure to test your code before you submit it.

For all functions make sure you've written good docstrings that include type contract, function description and the preconditions if any.

This assignment has three parts, the first about dictionaries and sets, the second one about objects and finally about Recursivity. Put all the required documents into a folder called a2\_XXXXXX where you changed XXXXXX to your student number, zip that folder and submit it in BrightSpace. In particular, the folder should have the following three files: a5\_part1\_XXXXXX.py, a5\_part2\_XXXXXX.py, a5\_part3\_XXXXXX.py and a5\_part2\_testing\_XXXXXX.txt where you changed XXXXXX to your student number.

For part 1, I provided you with starter code in file called a5\_part1\_XXXXXX.py. Begin by replacing XXXXXX in the file name with your student number. Then open the file. Your solution (code) for the assignment must go into that file in the clearly indicated spaces.

The file has main completely coded for you. Nothing else will go into the main. It also has some functions completely precoded for you. Your task will be to code the remaining functions. You are not allowed to delete or comment-out any parts of the provided code. The only exception to that rule is the keyword pass. Some functions have that keyword. You can remove it once you are done coding that function. You also must follow the instructions given in comments and implied by docstrings. You are however allowed to add your own additional (helper) functions. In fact **you must add at least THREE more function**.

If you are running out of ideas here are the names of some of the extra functions my solution has: remove\_punctuation(words), process\_lines(ls), make\_dict(lsw), is\_valid(D,query) is\_word(word) I have provided 5 text files to test and debug your code with as explained in the next section.

As always, you can make multiple submissions, but only the last submission before the deadline will be graded.

As always, each of your programs must run without syntax errors. In particular, when grading your assignment, TAs will first open your file a5\_part1\_XXXXXX.py with IDLE and press Run Module. The same will be done with a5\_part2\_XXXXXX.py. If pressing Run Module causes any syntax error, the grade for that part becomes zero. Furthermore, for each function whose code is missing, I have provided below one or more tests to test your functions with. To obtain a partial mark for these functions your solutions may not necessarily give the correct answer on these tests. But if your function gives any kind of Python error when run on the tests provided, that function will be marked with zero points. Finally, each function has to be documented with docstrings.

Using global variables inside of functions is not allowed. If you do not know what that means, for now, interpret this to mean that inside of your functions you can only use variables that are created in that function. For example, the following code fragment would not be allowed, since variable `x` is not a parameter of function `a_times(a)` nor is it a variable created in function `a_times(a)`. It is a global variable created outside of all functions.

```
def a_times(a):  
    result=x*a  
    return result
```

```
x=float(input("Give me a number: "))  
print(a_times(10))
```

## Part 1 Dictionaries and sets – 40 points

For this part, you will need to write a program that solves co-existence problem.  
What is co-existence problem?

You will write a Python program to solve the co-existence problem. The co-existence problem is stated as follows. We have a file containing English sentences, one sentence per line. Given a list of query words, your program should output the line number of lines that have all those words. While there are many ways to do this, the most efficient way is to use sets and dictionaries. Here is one example. Assume that the following is the content of the file. Line numbers are included for clarity; the actual file doesn't have the line numbers.

1. Try not to become a man of success, but rather try to become a man of value.
  2. Look deep into nature, and then you will understand everything better.
  3. The true sign of intelligence is not knowledge but imagination.
  4. We cannot solve our problems with the same thinking we used when we created them.
  5. Weakness of attitude becomes weakness of character.
  6. You can't blame gravity for falling in love.
  7. The difference between stupidity and genius is that genius has its limits.
- (These are attributed to Albert Einstein.)

If we are asked to find all the lines that contain this set of words: {"true", "knowledge", "imagination"} the answer will be line 3 because all three words appeared in line 3. If they appear in more than one line, your program should report all of them.

For example, co-existence of {"the", "is"} will be lines 3 and 7.

**IMPORTANT:** You should download a text file version of book War and Piece from from Gutenberg project. You can find it here under Plain Text UTF-8: <http://www.gutenberg.org/ebooks/2600>

Download the text version of that book and save it in the same directory as your program. Your solution should be instantaneous on that book, that is your program should produce the required dictionary in 1 or 2 seconds on that book and it should answer questions about any co-existence instantaneously.

Alternatively you can find the book here since I already downloaded it:

<https://www.dropbox.com/s/pg4p9snzv60rp5v/WarAndPiece.txt?dl=0>

## Python Implementation:

You need to implement the following functions:

### 1) open\_file()

The open\_file function will prompt the user for a file-name, and try to open that file. If the file exists, it will return the file object; otherwise it will re-prompt until it can successfully open the file. This feature must be implemented using a while loop, and a try-except clause.

### 2) read\_file(fp) This function has one parameter: a file object (such as the one returned by the open\_file() function).

This function will read the contents of that file line by line, process them and store them in a dictionary. The dictionary is returned. Consider the following string pre-processing:

1. Make everything lowercase
2. Split the line into words
3. Remove all punctuation, such as “,”, “.”, “!”, etc.
4. Remove apostrophes and hyphens, e.g. transform “can’t” into “cant” and “first-born” into “firstborn”
5. Remove the words that are not all alphabetic characters (do not remove “cant” because you have transformed it to “cant”, similarly for “firstborn”).
6. Remove the words with less than 2 characters, like “a”

Hint for string pre-processing mentioned above:

To find punctuation for removal you can import the string module and use string.punctuation which has all the punctuation.

To check for words with only alphabetic characters, use the isalpha() method. Furthermore, after pre-processing, you add the words into a dictionary with the key being the word and the value is a set of line numbers where this word has appeared.

For example, after processing the first line, your dictionary should look like:

```
{'try': {1}, 'not': {1}, 'to': {1}, 'become': {1}, 'man': {1}, 'of': {1}, 'success': {1}, 'but': {1}, 'rather': {1}, 'value': {1}}
```

This should be repeated for all the lines; the new keys are added to the dictionary, and if a key already exists, its value is updated. At the end of processing all these 7 lines, the value in the dictionary associated with key “the” will be the set {3, 4, 7}. (Note: the line numbers start from 1.)

3) find\_coexistence(D, query) The first parameter is the dictionary returned by read\_file; the second one is a string called query. This query contains zero or more words separated by white space. You need to split them into a list of words, and find the line numbers for each word. To do that, use the intersection or union operation on the sets from D (you need to figure out which operation is appropriate). Then convert the resulting set to a sorted list, and return the sorted list. (Hint: for the first word simply grab the set from D; for subsequent words you need to use the appropriate set operation: intersection or union.)

### 4) #main

The main part of the program should call the three functions above. Loop, prompting the user to enter space-separated words. Use that input to find the co-occurrence and print the results. Continue prompting for input until “q” or “Q” is inputted.

Very important considerations:

Every time you want to look up a key in a dictionary, first you need to make sure that the key exists. Otherwise it will result in an error. So, always use an if statement before looking up a key:

```
if key in data_dict:
```

```
    ## the key exists in a dictionary, so it is safe to use data_dict[key]
```

After you completed the program, see how it works for the two files provided: einstein.txt, and gettysburg.txt

## 2 Testing Part 1

### 2.1 Testing with einstein.txt file

Enter the name of the file: b.txt

There is no file with that name. Try again.

Enter the name of the file: grrrr

There is no file with that name. Try again.

Enter the name of the file: einstein.txt

Enter one or more words separated by spaces, or 'q' to quit: the

The one or more words you entered coexisted in the following lines of the file:

3 4 7

Enter one or more words separated by spaces, or 'q' to quit: the is

The one or more words you entered coexisted in the following lines of the file:

3 7

Enter one or more words separated by spaces, or 'q' to quit: true knowledge imagination

The one or more words you entered coexisted in the following lines of the file:

3

Enter one or more words separated by spaces, or 'q' to quit: bla

Word 'bla' not in the file.

Enter one or more words separated by spaces, or 'q' to quit: can't

The one or more words you entered coexisted in the following lines of the file:

6

Enter one or more words separated by spaces, or 'q' to quit:

Word " not in the file.

Enter one or more words separated by spaces, or 'q' to quit: ?

Word " not in the file

Enter one or more words separated by spaces, or 'q' to quit: a

Word 'a' not in the file.

Enter one or more words separated by spaces, or 'q' to quit: nature

The one or more words you entered coexisted in the following lines of the file:

2

Enter one or more words separated by spaces, or 'q' to quit: THE

The one or more words you entered coexisted in the following lines of the file:

3 4 7

Enter one or more words separated by spaces, or 'q' to quit: tHe

The one or more words you entered coexisted in the following lines of the file:

3 4 7

Enter one or more words separated by spaces, or 'q' to quit: q

More on test runs on file einstein.txt

RESTART: /Users/vidadujmovic/Dropbox/courses/python-iti1120-2019/assignments/assignment5/a5\_dict\_solution.py  
Enter the name of the file:

(Vida: Instead of a file name PRESS CTRL-C)

Then:

```
>>> f=open_file()
Enter the name of the file: ah.txt
There is no file with that name. Try again.
Enter the name of the file: einstein.txt
>>> f
<_io.TextIOWrapper name='einstein.txt' mode='r' encoding='UTF-8'>
>>> d=read_file(f)
>>> d
{'try': {1}, 'not': {1, 3}, 'to': {1}, 'become': {1}, 'man': {1}, 'of': {1, 3, 5}, 'success': {1}, 'but': {1, 3},
'rather': {1}, 'value': {1}, 'look': {2}, 'deep': {2}, 'into': {2}, 'nature': {2}, 'and': {2, 7}, 'then': {2},
'you': {2, 6}, 'will': {2}, 'understand': {2}, 'everything': {2}, 'better': {2}, 'the': {3, 4, 7}, 'true': {3},
'sign': {3}, 'intelligence': {3}, 'is': {3, 7}, 'knowledge': {3}, 'imagination': {3}, 'we': {4}, 'cannot': {4},
'solve': {4}, 'our': {4}, 'problems': {4}, 'with': {4}, 'same': {4}, 'thinking': {4}, 'used': {4}, 'when': {4},
'created': {4}, 'them': {4}, 'weakness': {5}, 'attitude': {5}, 'becomes': {5}, 'character': {5}, 'cant': {6},
'blame': {6}, 'gravity': {6}, 'for': {6}, 'falling': {6}, 'in': {6}, 'love': {6}, 'difference': {7},
'between': {7}, 'stupidity': {7}, 'genius': {7}, 'that': {7}, 'has': {7}, 'its': {7}, 'limits': {7}}
>>>
>>> find_coexistence(d, " the has")
[7]
>>> find_coexistence(d, " the is ")
[3, 7]
```

## 2.2 Testing with gettysburg.txt file

RESTART: /Users/vidadujmovic/Dropbox/courses/python-iti1120-2019/assignments/assignment5/a5\_dict\_solution.py  
Enter the name of the file: gettysburg.txt  
Enter one or more words separated by spaces, or 'q' to quit: nation  
The one or more words you entered coexisted in the following lines of the file:  
2 6 9 23  
Enter one or more words separated by spaces, or 'q' to quit: here dead  
The one or more words you entered coexisted in the following lines of the file:  
14 22  
Enter one or more words separated by spaces, or 'q' to quit: It is  
The one or more words you entered coexisted in the following lines of the file:  
10 17 19  
Enter one or more words separated by spaces, or 'q' to quit: 4you  
Word '4you' not in the file.  
Enter one or more words separated by spaces, or 'q' to quit: Q

## 2.3 Testing with WarAndPiece.txt file

>>>

RESTART: /Users/vidadujmovic/Dropbox/courses/python-iti1120-

2019/assignments/assignment5/a5\_dict\_solution.py

Enter the name of the file: WarAndPiece.txt

Enter one or more words separated by spaces, or 'q' to quit: hard life

The one or more words you entered coexisted in the following lines of the file:

33953 49922 60869

Enter one or more words separated by spaces, or 'q' to quit: 2013

Word '2013' not in the file.

Enter one or more words separated by spaces, or 'q' to quit: VIII

The one or more words you entered coexisted in the following lines of the file:

52 110 154 194 228 274 328 356 402 450 530 600 634 674 714 756 790

2079 8264 13577 17689 20153 23726 27877

30215 33840 38274 45012 51021 53356 55805 58145 61010 63871

Enter one or more words separated by spaces, or 'q' to quit: black-eyed

The one or more words you entered coexisted in the following lines of the file:

2682 49686 61292

Enter one or more words separated by spaces, or 'q' to quit: black-eyed wide-mouthed

The one or more words you entered coexisted in the following lines of the file:

2682

Enter one or more words separated by spaces, or 'q' to quit: What's the good of denying it, my dear?

The one or more words you entered coexisted in the following lines of the file:

2900

Enter one or more words separated by spaces, or 'q' to quit: q

## Part 2 Objects – 40 points

For this part, you are provided with 3 files: [a5\\_part2\\_XXXXXX.py](#), [a5\\_part2\\_testing\\_given.txt](#) and [drawings\\_part2.pdf](#)

File [a5\\_part2\\_XXXXXX.py](#) already contains a class [Point](#) that we developed in class. For this part, you will need to develop and add two more classes to [a5\\_part2\\_XXXXXX.py](#): class [Rectangle](#) and class [Canvas](#).

To understand how they should be designed and how they should behave, you must study in detail the test cases provided in [a3\\_part2\\_testing\\_given.txt](#). These tests are your main resource in understanding what methods your two classes should have and what their input parameters are. I will explain few methods below in detail, but only those whose behaviour may not be clear from the test cases. You will need to also submit your own text file called [a5\\_part2\\_testing\\_XXXXXX.txt](#) demonstrating that you tested your two classes and their methods (in particular, demonstrating that you tested them by running all the calls made in [a5\\_part2\\_testing\\_given.txt](#))

=====  
Details about the two classes:  
=====

**Class Rectangle** represents a 2D (axis-parallel) rectangle that a user can draw on a computer screen. Think of a computer screen as a plane where each position has an x and a y coordinate.

The data (i.e. attributes) that each object of type Rectangle should have (and that should be initialized in the constructor, i.e., `__init__` method of the class Rectangle) are:

- \* **two Points**: the first point representing the bottom left corner of the rectangle and the second representing the top right corner of the rectangle; and,
- \* the **color** of the rectangle

Note that the two points (bottom left and top right) completely determine (the axis parallel) rectangle and its position in the plane. There is no default rectangle.

(see [drawings\\_part2.pdf](#) file for some helpful illustrations)

The `__init__` method of Rectangle (that is called by the constructor Rectangle) will take two objects of type Point as input and a string for the color). You may assume that the first Point (passed to the constructor, i.e. `__init__`) will always have smaller than or equal x coordinate than the x coordinate of the second Point and smaller than or equal y coordinate than the y coordinate of the second Point.

**Class Rectangle** should have **13 methods**. In particular, in addition to the constructor (i.e. `__init__` method) and **three methods** that override python's object methods (and make your class user friendly as suggested by the test cases), your class should contain the following 9 methods:

`get_bottom_left`, `get_top_right`, `get_color`, `reset_color`, `get_perimeter`, `get_area`, `move`, `intersects`, and `contains`.

Here is a description of three of those methods whose job may not be obvious from the test cases.

\* Method **move**: given numbers dx and dy this method moves the calling rectangle by dx in the x direction and by dy in the y-direction. This method should not change directly the coordinates of the two corners of the calling rectangle, but must instead call move method from the Point class.

\* Method **intersects**: returns True if the calling rectangle intersects the given rectangle and False otherwise. Definition: two rectangles intersect if they have at least one point in common, otherwise they do not intersect.

\* Method **contains**: given an x and a y coordinate of a point, this method tests if that point is inside of the calling rectangle. If yes it returns True and otherwise False. (A point on the boundary of the rectangle is considered to be inside).

---

**Class Canvas** represents a collection of Rectangles. It has **8 methods**. In addition, to the constructor (i.e. `__init__` method) and **two methods** that override python's object methods (and make your class user friendly as suggested by the test cases), your class should contain 5 more methods: `add_one_rectangle`, `count_same_color`, `total_perimeter`, `min_enclosing_rectangle`, and `common_point`.

Here is a description of those methods whose job may not be obvious from the test cases.

\* The method **total\_perimeter**: returns the sum of the perimeters of all the rectangles in the calling canvas. To compute total perimeter do not compute a perimeter of an individual rectangle in the body of total\_perimeter method. Instead use get\_perimeter method from the Rectangle class.

\* Method **min\_enclosing\_rectangle**: calculates the minimum enclosing rectangle that contains all the rectangles in the calling canvas. It returns an object of type Rectangle of any color you prefer. To find minimum enclosing rectangle you will need to find the minimum x coordinate of all rectangles, the maximum x coordinate for all rectangles, the minimum y coordinate and the maximum y coordinate of all rectangles.

\* Method **common\_point**: returns True if there exists a point that intersects all rectangles in the calling canvas. To test this (for axis parallel rectangles like ours), it is enough to test if every pair of rectangles intersects (according to a Helly's theorem for axis-aligned rectangles: [http://en.wikipedia.org/wiki/Helly's\\_theorem](http://en.wikipedia.org/wiki/Helly's_theorem) ).

\*\*\*

Finally recall, from the beginning of the description of this assignment that each of your methods should have a **type contract**.

### Part 3 Recursion – 20 points

- a. Derive a **recursive** Python function `stars` that will take as a parameter a non negative integer and will generate a drawing composed of stars as displayed below. You can use a loop to generate a line of stars, but not the entire drawing.

Example :

```
>>> stars(4)
****
***
**
*
*
**
***
****
```

- b. Derive **recursive** Python function called `sumListPos_rec` that will take as parameter a list and as a second parameter the number of the list elements, and that will return the sum of the positive elements ( $> 0$ ).

Example :

```
>>> l = [1,-2,5,0,-5]
>>> sumListPos_rec(l, len(l))
6
```

**Note:** If the functions in both a and b are not recursive, the number of points received will be **zero**.