Object-Oriented Programming
50:198:113 (Fall 2022)

Homework:	**1**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**10/11/22**	**URL:**	`https://sites.rutgers.edu/suneeta-ramaswami`
Office:	**321 BSB**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
		Phone:	**(856)-225-6439**

Homework Assignment 2

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student's own work. You may seek or accept assistance only from the course instructor. Any violation of this rule will be dealt with harshly.

This assignment is primarily a review of file processing, dictionaries, loops, strings, functions, and writing modular programs with minimal code repetition. In this and all future assignments, you are graded not only on the correctness of the code, but also on clarity and readability. Hence, I will deduct points for poor indentation, poor choice of object names, and lack of documentation. All modules and function definitions must use docstring (triple quote) documentation. For the remaining code, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented *briefly* in clear English.

Please read the submission guidelines at the end of this document before you start your work.

Important note: When writing each of the following programs, it is important that you name the functions exactly as described because I will assume you are doing so when testing your programs. If your program produces errors because the functions do not satisfy the stated prototype, points will be deducted. If I am unable to import your module due to syntax errors, I will automatically deduct 50% of the points for that problem. You may implement additional helper functions if you wish, but you must, at a minimum, implement the functions you are asked to in each of the two problems. In particular,

1. For Problem 1, download the module `problem1.py` from Canvas. This file contains test code for your function implementations. You should insert your implementation for Problem 1 into this file.
2. For Problem 2, implement both functions in a module called `problem2.py`.

Problem 1 [30 points] Polynomials. In this problem, you are asked to implement *polynomials* using a Python dictionary. A polynomial is an expression of the form

$$c_0 + c_1x + c_2x^2 + c_3x^3 + \ldots + c_nx^n$$

where $c_0, c_1, \ldots, c_n$ are real numbers. Each of c_ix^i is called a *term* of the polynomial, and c_i is called the *coefficient* of the term with *exponent* i. Note that c_0 is simply the coefficient of the term with exponent 0. The maximum exponent with a non-zero coefficient is called the *degree* of the polynomial. For example, $6x^{14} + 9x^{11} - 12x^3 + 42$ is a polynomial of degree 14. The polynomial $-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9$ has degree 6. The following are standard operations on polynomials:

Scaling a polynomial: Given a polynomial $p(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$ and a real value s, *scaling* $p(x)$ by s gives the polynomial $s \cdot p(x)$ obtained by scaling the coefficient of every term in $p(x)$ by the factor s. For example, if $p(x) = 6x^{14} + 9x^{11} - 12x^3 + 42$, then $2p(x) = 12x^{14} + 18x^{11} - 24x^3 + 84$.

Sum of two polynomials: The sum of two polynomials $p_1(x)$ and $p_2(x)$, denoted by $p_1(x) + p_2(x)$, is the polynomial obtained by adding the terms of $p_1(x)$ and $p_2(x)$. For example, if $p_1(x) = 6x^5 - 4x^3 + 10x^2 + 4$ and $p_2(x) = 3x^9 - 5x^7 - 3x^2 + 8x$, then $p_1(x) + p_2(x) = 3x^9 - 5x^7 + 6x^5 - 4x^3 + 7x^2 + 8x + 4$.

Difference of two polynomials: The difference of two polynomials $p_1(x)$ and $p_2(x)$ is the polynomial obtained by subtracting one from the other. Therefore, $p_1(x) - p_2(x)$ is the polynomial obtained by subtracting the terms of $p_2(x)$ from $p_1(x)$. For example, if $p_1(x) = 6x^5 - 4x^3 + 10x^2 + 4$ and $p_2(x) = 3x^9 - 5x^7 - 3x^2 + 8x$, then $p_1(x) - p_2(x) = -3x^9 + 5x^7 + 6x^5 - 4x^3 + 13x^2 - 8x + 4$.

Product of two polynomials: The product of two polynomials $p_1(x)$ and $p_2(x)$, denoted by $p_1(x) \cdot p_2(x)$, is the polynomial obtained by the pair-wise multiplication of terms in $p_1(x)$ and $p_2(x)$. For example, if $p_1(x) = x^4 + 4x^3 + 4x^2$ and $p_2(x) = 2x - 1$, then $p_1(x) \cdot p_2(x) = 2x^5 + 7x^4 + 4x^3 - 4x^2$. This is obtained as follows: $(x^4 + 4x^3 + 4x^2)(2x) + (x^4 + 4x^3 + 4x^2)(-1) = 2x^5 + 8x^4 + 8x^3 - x^4 - 4x^3 - 4x^2 = 2x^5 + 7x^4 + 4x^3 - 4x^2$.

In this program, you will store a polynomial in a dictionary. Recall that a dictionary stores key-value pairs as key:value. For a polynomial, each term forms a key:value pair: the exponent of the term is the key, and the coefficient of the term is the value. *(Be careful not to do it the other way around in your code!)* The keys are integers and the values are real numbers. For example, the polynomial $6x^{14} + 9x^{11} - 12x^3 + 42$ will be stored as the dictionary {`14:6, 11:9, 3:-12, 0:42`}. The polynomial $-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9$ will be stored as the dictionary {`6:-12, 5:5, 4:-20, 2:8, 1:-12, 0:9`}.

You are asked to implement the following functions to manipulate polynomials:

1. A function called `read_polynomial` with a single parameter: a string `filename`. The function should open the file called `filename` and use it for reading. The file stores the terms of a polynomial as follows: It has as many lines as there are terms, and each line in the file contains a pair of numbers: the first number is the exponent of the term and the second one is the coefficient of the term. Your function should read in the polynomial from the file and store it in a dictionary (as described above). *The function should* `return` *the dictionary.* Remember to close the file after you have read it.

   Two files called `poly1.txt` and `poly2.txt` containing polynomial data have been created for you. They are available on Canvas along with the homework assignment. You may download them to test your program. You should also create some of your own files for further testing.

2. A function called `print_polynomial` with a single parameter: a polynomial `P`. Keep in mind that `P` is a dictionary. This function prints out the polynomial. The polynomial must be printed "nicely". This means that only terms with non-zero coefficients must be printed out and the terms must be printed from largest exponent to the smallest. Furthermore, if the coefficient of a term is negative, the `-` (minus sign) must be embedded in the printed polynomial. For example, if the polynomial is $-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9$, it should be printed as

```
-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9
```

and **not** as

```
-12x^6 + 5x^5 + -20x^4 + 0x^3 + 8x^2 + -12x + 9
```

3. A function called `degree` with a single parameter: a polynomial `P`. The function should `return` the degree of `P`.
4. A function called `eval_polynomial` with two parameters: a polynomial `P` and a real number `X`. The function should evaluate `P` at $x =$ `X` and `return` the result. For example, if `P` is the polynomial $-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9$, then `eval_polynomial(P, 2)` should return -911 (which we obtain by plugging in 2 as the value of x).
5. A function called `addterm` with three parameters: a polynomial `P`, an exponent `exp`, and a coefficient `coeff`. This functions adds to the polynomial `P` a term with exponent `exp` and coefficient `coeff`. If a term with this exponent already exists, the new term is simply added on to it. For example, if `P` is the polynomial $-4x^3 + 13x^2 + 4$, then `addterm(P, 5, 6)` adds the term $6x^5$ to `P`, so that `P` is now $6x^5 - 4x^3 + 13x^2 + 4$. Subsequently, `addterm(P, 2, -3)` causes `P` to become $6x^5 - 4x^3 + 10x^2 + 4$. *Note:* If the coefficient of a term becomes 0 as a result of adding the new term, it should be deleted from the polynomial.
6. A function called `removeterm` with two parameters: a polynomial `P` and an exponent `exp`. The function removes from the polynomial `P` the term with exponent `exp`. If such a term does not exist, the polynomial remains unchanged.
7. A function called `scale_polynomial` with two parameters: a polynomial `P` and a real value `s`. The function should `return` the polynomial `sP`, that is, the polynomial obtained by scaling `P` by `s`.
8. A function called `sum_polynomial` with two parameters: a polynomial `P1` and a polynomial `P2`. The function should `return` the sum polynomial obtained by adding `P1` and `P2`. *Hint:* Use the function `addterm` to create the sum polynomial; do not repeat code unnecessarily.
9. A function called `diff_polynomial` with two parameters: a polynomial `P1` and a polynomial `P2`. The function should `return` the difference polynomial obtained by subtracting `P2` from `P1`. *Hint:* The function `addterm` can be used here as well.
10. A function called `prod_polynomial` with two parameters: a polynomial `P1` and a polynomial `P2`. The function should `return` the product polynomial obtained by multiplying `P1` and `P2`. *Hint:* Once again, use the function `addterm` to create the product polynomial.
11. A function called `test_polynomials` without any parameters. **This function has been written for you in a file called** `problem1.py`, available for download on Canvas. This function is simply a driver function that makes calls to all of the above functions, thus allowing you to test them.

Download the following files from the Canvas site for this assignment: `problem1.py`**, and the sample data files** `poly1.txt` **and** `poly2.txt`**.**

Problem 2 [30 points] Document similarity. Several text processing applications require an analysis of the word frequency distribution in the text. The term *word frequency* refers to the number of occurrences of each word in the text. For example, to compare if two documents are similar to each other, there are various algorithms that require word frequency information about the text contained in the documents. (This is how websites recommend news articles that are similar to ones you have read previously, for instance.)

In this problem, you will first find the frequency distribution of words in a given text file, i.e., the number of times each unique word occurs in the file. **You are asked to use a dictionary** to store the frequency distribution of the words. Furthermore, you will compare two documents for similarity by using their word frequency distribution. Additional details are provided below:

- For our purposes, a word is simply a consecutive sequence of characters that are letters of the alphabet. We do not distinguish between upper and lower case (e.g., "The" and "the" are the same word). For example, the text "How old are you and how old is your half-brother?" consists of nine words: "how", "old", "are", "you", "and", "is", "your", "half", and "brother". The words "how" and "old" have a frequency of 2 each, and the remaining words have a frequency of 1 each.

  Note that we do not include punctuation characters in a word. This is why, in the example above, the word "half-brother" is treated as two words, "half" and "brother". This also means that words such as "Sally's" or "They're" get treated as two words each ("Sally" and 's', "They" and "re"), but we'll live with that. There are various ways in which you can split a word at its punctuation marks. For example, you can use the `replace()` method for strings to replace all the punctuation marks with a space (make sure you do not replace it with an empty string because that will combine hyphenated words into one word, for example). In the `string` module, you can get the string of all punctuation characters by using `string.punctuation`. You may use this if necessary.

- To compare two documents to check if their contents are similar to each other, we will use a commonly used method called *cosine similarity* that uses their word frequency distribution. This method works as follows: Let $w_1, w_2, w_3, \ldots, w_n$ be the set of n unique words from both the documents. Some of these words will occur in only one document and some will occur in both. Let $(a_1, a_2, a_3, \ldots, a_n)$ be the frequency of these words in the first document (hence, if a word w_i does not occur in the document, a_i will be 0). Let $(b_1, b_2, b_3, \ldots, b_n)$ be the frequency of the words in the second document (similarly, some of the b_i may be 0). Then the cosine similarity between the two documents is defined as follows:

$$\frac{a_1.b_1 + a_2.b_2 + a_3.b_3 + \ldots + a_n.b_n}{\sqrt{a_1^2 + a_2^2 + a_3^2 + \ldots + a_n^2}\sqrt{b_1^2 + b_2^2 + b_3^2 + \ldots + b_n^2}}$$

  (For those of you that know about vectors, note that the above expression is simply the cosine of the angle between two n-dimensional vectors.) The above expression will give a value between 0 and 1. The closer the above value is to 1, the more similar the two documents, and the closer it is to 0, the more dissimilar. Here is an example:

  - Suppose the first document contains: `starry starry night`
  - Suppose the second document contains: `a clear night is starry`
  - Then the set of unique words in the documents is `starry, night, a, clear, is`. The frequency of these words in the first document is $(2, 1, 0, 0, 0)$ and in the second document is $(1, 1, 1, 1, 1)$. The cosine similarity between them is $3/(\sqrt{5}\sqrt{5}) = 3/5 = 0.6$. We can conclude that the two documents are somewhat similar.

As it turns out, this method has the weakness that some commonly occuring words (such as *a, an, the, and, of, to*, etc.) can have an outsize influence on determining similarity. Most documents have a lot of these words, so they should not play much of a role in determining whether two documents are similar or not. There are many algorithms to deal with this issue, but we will ignore it in our program here.

Implement the following functions for this problem:

1. (*15 points*) A function called `freq_dictionary` with a single parameter `infile`, the name of the text file for which we want to compute the word frequency distribution. The function should first open `infile` for reading. It should then create a dictionary with key:value pairs, where the key is a word occurring in `infile` and its associated value is the frequency of that word in the file i.e., the number of times it occurs in the file. Refer to the earlier discussion for our definition of a word. Note that every word occurring in the file should be included in the dictionary (even if they are nonsense words). Words that do not occur in the file should not be included in the dictionary. The function must **return** the dictionary. Remember to close the file *prior to* the return statement.

2. (*15 points*) A function called `cosine_similarity` with two parameters: `docfile1` and `docfile2`, which are names of two text files that we want to compare for similarity. The function should open both files for reading, create their word frequency dictionaries, and return their cosine similarity (a real number). Remember to close both files before returning. *Hint:* Make good use of the `get` method for dictionaries to implement this function in a clean and simple way.

Submission Guidelines

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Canvas as follows:

1. Go to the "Assignments" tab of the Canvas site for this course.
2. Click on "Programming Assignment 2" under Homework Assignments.
3. Download this homework document (`hw2.pdf`). Carefully read the problem descriptions in the homework document before creating the module for each problem.
4. Also download the module `problem1.py`, the data files `poly1.txt, poly2.txt` for Problem 1, and the sample documents to test your code for Problem 2.
5. Use this same link to upload your homework files (`problem1.py` and `problem2.py` when you are ready to submit. Please make sure your name appears in the docstring documentation in each module.

You must submit your assignment at or before 11:59PM on October 11, 2022.