Laboratory Assignment 5

Problem 1 - nth frequency `[5 pts]`

Write a function that takes two arguments; a list and an integer called n. Return the list of numbers that repeats n times.

- Function parameter names do not matter, but the function should expect 2 parameters.
- `inputs :`
  - `l = { x : x is list and len(x) ∈ [0:200]} and` $l_i \in \mathbb{Z}$
  - $n \in \mathbb{Z}$
- `output : list`

```
>>> problem1([1, 2, 1, 1, 1, 2, 3, 4], 1)
[3, 4]

>>> problem1([1, 2, 1, 1, 1, 2, 3, 4], 0)
[]

>>> problem1([1, 2, 1, 1, 1, 2, 3, 4], -2)
[]

>>> problem1([1, 2, 1, 1, 1, 2, 3, 4], 2)
[2]

>>> problem1([1, 2, 1, 1, 1, 2, 3, 4], 3)
[]

>>> problem1([], 0)
[]

>>> problem1([], 1)
[]
```

Problem 2 - Median grade `[7 pts]`

Write a function that takes one argument: a dictionary. It will have the *name, grade* pair for the INF211 class. Find and return the median grade of the class.

- If the number of people is odd, return the middle grade.
- If the number of people is even, take the mean (average) of the two middlemost numbers and return that.
- if the dictionary is empty, return 0.
- Function parameter names do not matter.
- `input : l = { x : x is dict and len(x) ∈ [0:1000]}`
- `output : integer or float`

```
>>> problem2({'ahmet': 30, 'ali': 20, 'mehmet':10})
20

>>> problem2({'ahmet': 10, 'ali': 20})
15
```

Problem 3 - Read Transcript `[13 pts]`

Write a function that takes a filename as parameter. File will have a list of courses with the credits , given term and the scores. You need to read this file, create a dictionary for each course and add it to a list. Finally return this list. Example file is given with the assignment.

- Function parameter names do not matter.
- Dictionary keys will be: ***name***, ***credit***, ***term,*** and ***grade***.
- If the given course does not have a grade, enter **NA.**
    - There will be a comma at the end for sure. Check the example txt file.
- File will only have `'\n'` for the newline.
- File will use a comma as a separator between items.
- File will have ***name***, ***credit***, ***term***, ***grade*** in that order.
- If the file is empty, return an empty list.
- If the file cannot be read or not found or not properly parsed, return an empty list.
- `input : f = filename as string`
- `output : list`

```
>>> problem3(transcript1.txt')
[{'name' : 'inf211', 'credit' : 6, 'term' : 1, 'grade' : 'DC'},
 {'name' : 'elm101', 'credit' : 2, 'term' : 1, 'grade' : 'BA'},
 {'name' : 'mat101', 'credit' : 7, 'term' : 1, 'grade' : 'BB'}]

>>> problem3(transcript2.txt')
[{'name' : 'inf211', 'credit' : 6, 'term' : 1, 'grade' : 'DC'},
 {'name' : 'elm101', 'credit' : 2, 'term' : 1, 'grade' : 'BA'},
 {'name' : 'mat101', 'credit' : 7, 'term' : 1, 'grade' : 'BB'},
 {'name' : 'inf212', 'credit' : 6, 'term' : 2, 'grade' : 'NA'}]

>>> problem3('transcript_empty.txt')
[]

>>> problem3('some_random_filename_that_does_not_exist.txt')
[]
```

Problem 4 - Term GPA `[14 pts]`

Write a function that takes two parameters: a list that holds the transcript as dictionary items (as given in the problem 3), and a term number. You need to calculate the GPA for a given term and return the term GPA.

- Function parameter names do not matter.

- If any of the courses in the given term does not have a grade, do not take that into calculation.
- If the dictionary is empty, or the term does not have any courses, return `0`.
- Each grade corresponds to a number:
    - `AA - 4.0, BA - 3.5, BB - 3.0, CB - 2.5, CC - 2.0, DC - 1.5, DD - 1.0, FF - 0.0, NA - NA`
    - Note that `FF` will be taken into consideration but `NA` will not be taken into consideration.
- GPA calculation is done using weighted average.
- `input :`
    - `d = list of dictionaries`
    - `t = { x : x ∈ [1:8]}`
- `output : float`

```
>>> problem4(
[{'name' : 'inf211', 'credit' : 6, 'term' : 1, 'grade' : 'DC'},
 {'name' : 'elm101', 'credit' : 2, 'term' : 1, 'grade' : 'BA'},
 {'name' : 'mat101', 'credit' : 7, 'term' : 1, 'grade' : 'BB'}], 1)
2.46

>>> problem4(
[{'name' : 'inf211', 'credit' : 6, 'term' : 1, 'grade' : 'DC'},
 {'name' : 'elm101', 'credit' : 2, 'term' : 1, 'grade' : 'BA'},
 {'name' : 'mat101', 'credit' : 7, 'term' : 1, 'grade' : 'NA'}], 1)
2.0

>>> problem4(
[{'name' : 'inf211', 'credit' : 6, 'term' : 1, 'grade' : 'DC'},
 {'name' : 'elm101', 'credit' : 2, 'term' : 1, 'grade' : 'BA'},
 {'name' : 'mat101', 'credit' : 7, 'term' : 1, 'grade' : 'FF'}], 1)
1.06

>>> problem4(
[{'name' : 'inf211', 'credit' : 6, 'term' : 1, 'grade' : 'DC'},
 {'name' : 'elm101', 'credit' : 2, 'term' : 1, 'grade' : 'BA'},
 {'name' : 'mat101', 'credit' : 7, 'term' : 1, 'grade' : 'BB'}], 2)
0

>>> problem4([], 1)
0
```

Problem 5 - Black Box Testing `[16 pts]`

Write a function that takes two parameters: a function, and a number. This function's definition is given below, and the function may or may not work as expected. Your function should figure out if the given function is working or not working as expected. Return True if the function works correctly, False otherwise.

- Function parameter names do not matter.
- The function that will be tested takes one parameter, n, and should **find and return the number of 1's up to (including) the given number starting from 0.**
    - For example
        - ffunc(10) should return 2 since there are two 1's (1, and 10)
        - ffunc(100) should return 21 since there are 21 1's (1, 10, 11, 12, ..., 19, 21, 31, 41, ... 91, 100)
- `inputs :`
    - `ffunc = function that takes one parameter and returns the number of n's up to the given number (including) starting from 0.`
    - `n ∈ [0:300]`
- `output : Boolean`

```
>>> problem5(ffunc, 10)
True

>>> problem5(ffunc, 10)
False
```

Explanation: `if ffunc implementation is correct (you need to test this to make sure it is) returns True, if ffunc implementation is incorrect, returns False.`

Problem 6 - Combinations [10 pts]

Write a program that accepts a string and returns all possible combinations of the given strings.

- There should not be any duplicates in the returned list.
- The list should be ordered.
- Returned list should be lowercase.
- The words do not need to have a meaning.
- `inputs : s = { x : x is English letters and len(x) ∈ [1:10]}`
- `outputs: list of strings`

```
>>> problem6("a")
['a']

>>> problem6("AB")
['a', 'ab', 'b', 'ba']

>>> problem6("aba")
['a', 'aa', 'aab', 'ab', 'aba', 'b', 'ba', 'baa']
```

Problem 7 - Anagrams [5 pts]

Write a function that takes two parameters. First parameter is a string of arbitrary length, and the second parameter is a filename for a file that holds valid words. Write the function so that it should find all possible combinations of this string that are included in the given words list. Return the valid words, all lowercase and sorted, as a list.

- Function parameter names do not matter.
- For testing words, you can generate your own, or you can download [1].
- If no words are found, return an empty list.
- if an empty string is given, return an empty list.
- `inputs :`
  - `s : s = { x : x is English letters and len(x) ∈ [1:20]}`
  - `f : filename`
- `output : list of strings`

```
>>> problem7("abaa", "words.txt")
['a', 'aa', 'aaa', 'ab', 'aba', 'b', 'ba', 'baa']

>>> problem7("ABAA", "words.txt")
['a', 'aa', 'aaa', 'ab', 'aba', 'b', 'ba', 'baa']
```

Problem 8 - Sub-matrix [14 pts]

Write a function that will take two parameters, both are list of lists representing a matrix. The function should find and return True if the second matrix is a sub-matrix of the first one. If it is not a sub-matrix, return False.

- Function parameter names do not matter.
- For example, for an m matrix given as :

  ```
  1 2 3
  4 5 6
  7 8 9
  ```

  Some of the valid sub-matrices (n) are given as:

  ```
  1 2 3     1 2 3     2    1    1 2 3     1 2     1    5 6
  4 5 6               5         4 5 6     4 5     4
  7 8 9               8                           7
  ```

- inputs :
  - m = { x : x is list } and m_i = { x : x is list } and **len(m) ∈ [1:10]** and **len(m_i) ∈ [1:10]** and $m_{ij} \in \mathbb{Z}$
  - n = { x : x is list } and n_i = { x : x is list } and **len(n) ∈ [1:10]** and **len(m_i) ∈ [1:10]** and $n_{ij} \in \mathbb{Z}$
- output : Boolean

```
>>> problem8([[1, 2], [3, 4]], [[1]])
True

>>> problem8([[1, 2], [3, 4]], [[5]])
False
```

```
>>> problem8([[1, 2], [3, 4]], [[1, 2], [3, 4]])
True

>>> problem8([[1, 2], [3, 4]], [[1, 3]])
False

>>> problem8([[1, 2], [3, 4]], [[1], [3]])
True

>>> problem8([[1, 2], [3, 4]], [[1, 2, 3], [3, 4, 5]])
False
```

Problem 9 - Compression `[7 pts]`

Our engineers developed a novel compression algorithm that they claim will save a lot of space when saving text files. In this algorithm, if a letter is consequently repeated more than 1 time, instead of writing the letter multiple times, a number is appended to denote how many times the preceding letter is repeated. Write a program that compresses the given string, and returns the compressed version as a string and the **compression rate** as a percentage **round**ed to an integer.

- Function parameter names do not matter.
- **Compression rate** is described as the percentage of saved space.
- If the letter happens only 1 time, no number should be appended.
- `inputs : s = { x : x is English letters and len(x) ∈ [0:1000]}`
- `outputs: a tuple of (string, integer)`

```
>>> problem9('gol')
'gol', 0

>>> problem9('golllllll')
'gol8', 60

>>> problem9('ggoooooooooooolllllllllllll')
'g2o12l13', 70

>>> problem9('ggoooooooooooollllllllllllll')
'g2o12l14', 71
```

Problem 10 - Missing Number `[9 pts]`

Write a function that will find the missing number in a given list of numbers. The numbers in the list are consecutive but might not be sorted. If no number is missing, return the next number.

- Function parameter names do not matter.
- There will always be either 1, or no missing numbers.
- `inputs : a = { x : x is list of integers and len(x) ∈ [0:200]}`

- `output : integer`

```
>>> problem10([1, 3, 2, 5])
4

>>> problem10([4, 1, 2])
3

>>> problem10([0, -1, -2, 2, 3])
1

>>> problem10([0])
1

>>> problem10([5, 3, 4])
6
```

[1] https://raw.githubusercontent.com/dwyl/english-words/master/words_alpha.txt