University of Essex

School of Computer Science and Electronic Engineering

CE705: Introduction to Programming in Python

Set by: Dr. Morteza Varasteh
Due Date: 17 January 2023 by 12:00 am (UK time) via FASER.
Assignment maximum mark: 100
Proportion to final module mark: 40%

SUBMISSION REQUIREMENTS:
Students are required to submit onlyONE single .py file containing the codes for this assignment no later than 12:00 am (UK time) on the 17/01/2023. The standard lateness penalty will be applied to late work. Do NOT include any project file (i.e., no files other than the .py).

FEEDBACK FROM THIS ASSIGNMENT
Individual feedback will be provided within 4 weeks of the due date.

F.A.Q. (must be read)

1) Can I submit anything other than a working .py file?
Absolutely not. You must submit a .py file that works "out-of-the-box" in Idle. If you submit anything else (e.g. ipynb, pyw, pyi, etc.) this will be an automatic fail. You must submit a .py file that works in Idle. The name of the .py file must be your student ID number. For example, if your ID number is 19974, your submitted file must be 19974.py

2) Can I import modules?
You can import any modules that come with Python (e.g., math, os, etc). You cannot use any module that requires extra installation (e.g., Pandas). The only exception to this rule is NumPy.

3) Can I make a small change in the return type of a function or method?
No. If a function or method is supposed to return a number, say 5, and you return '5', [5] or anything other than just 5, you will lose all marks related to this function or method.

4) Can I make a small change in the data type of a parameter?
No. Such changes will lead to you losing all marks related to the function or method in question.

5) Can I add or remove a parameter?
No. Such changes will lead to you losing all marks related to the function or method in question.

6) Can I make a minor change in the name of a function or method?
No. Such changes will lead to you losing all marks related to the function or method in question. Please note that Python is case-sensitive. For instance, the name `run_test` is not the same thing as `Run_Test`. So please use the given names in the exercises for your functions, methods, or classes.

7) Can I implement extra functions or methods to make my code easier/cleaner?
Yes. Please note you must implement all the functions and methods described in the assignment brief. If you'd like to implement more, you are welcome to do so.

8) Can I implement the algorithm in this assignment in any other way than what the assignment brief describes?
Yes. However, the full mark will be granted to those who follow the guidelines given as 'Hint' in each exercise. In case there is no hint provided, feel free to write the solution in any way you like.

9) Why am I not allowed to make arbitrary changes?
Large pieces of software (e.g. Windows) are not written by a single programmer, but by many. All programmers will be working on different parts of the software, but all of these parts are likely to interact in some way. The programme specification makes sure everybody knows what each function expects to receive and what each function should return. If one programmer unilaterally decides to make a small change that goes against the specification… then the software will not work as expected.

Marking Scheme

Characteristics of an excellent project (70% or more):
• Excellent code documentation
• Excellent use of Python's native methods and code standards
• Excellent use of relevant data types
• Follows carefully the specification provided
• Implements the described run_test, which shows the expected results.
• Excellent code optimisation in terms of memory, speed and readability
• Generally, an excellent solution, carefully worked out.

Characteristics of a good project (60%):
• Good code documentation
• Good use of Python's code standards
• Good use of relevant data types
• Follows the specification provided, with no major deviations.
• Implements the described run_test, which shows the expected results.
• Good code optimisation in terms of memory, speed and readability
• Generally, a good solution, which delivers what the final user would expect.

Characteristics of a fair project (50% or less):
• No meaningful code documentation
• Code tends to be more verbose than actually needed or at times difficult to read
• No real thought on the relevance of data types
• Does not follow the specification provided (this alone will indicate a fail).
• It contains code that is outside of a function or method
• It is not a .py file that runs in Idle (this alone will indicate a fail)
• It keeps printing things on the screen (run_test is an exception to this).
• Does not implement run_test as described, or this does not show the expected results.
• A solution that only seems to deliver what the final user would expect.

Please note:
• You must submit only one file
• You must follow the instructions for each function and method in terms of parameters and returns
• You should document your code, i.e., proper commenting, easy to follow, sensible variable names, proper usage of commands and avoiding redundant commands and variables.
• It is a good idea to test each function/method at a time, and only afterwards test the
whole project

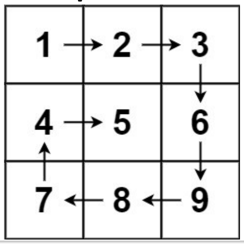
Question 1) You are given a list of numbers `num`. Write a function `Exercise1(num)` returning a list `ans`, such that `ans[i]` is equal to the product of all the numbers of `ans` except `ans[i]`. For example, if your input is `num = [1,2,3,4]`, your function should return `[24, 12, 8, 6]`.
Note that to get the full mark, you are <u>only allowed</u> to use the built-in functions `len()`, `range()` and maybe `reversed()`.
Mark: [10%]

Hint:

1. Initialize two empty arrays, `L` and `R` where for a given index `i`, `L[i]` would contain the product of all the numbers to the left of `i` and `R[i]` would contain the product of all the numbers to the right of `i`.
2. For the array `L`, $L[0]$ would be `1` since there are no elements to the left of the first element. For the rest of the elements, we simply use
$$\mathbf{L}[\mathbf{i}] = \mathbf{L}[\mathbf{i} - \mathbf{1}] * \mathbf{num}[\mathbf{i} - \mathbf{1}].$$
3. For the other array, we do the same thing but in reverse i.e., we start with the initial value of `1` in $R[\text{length} - 1]$ where length is the number of elements in the list and keep updating `R[i]` in reverse. Essentially,
$$\mathbf{R}[\mathbf{i}] = \mathbf{R}[\mathbf{i} + \mathbf{1}] * \mathbf{num}[\mathbf{i} + \mathbf{1}]$$
4. Once we have the two arrays set up properly, we simply iterate over the input array one element at a time, and for each element at index `i`, we find the wanted result by calculating $L[i] * R[i]$.

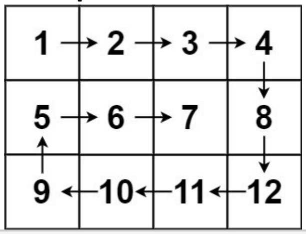
Question 2) You are given an $m \times n$ `matrix`. Write a function `Exercise2(matrix)` that returns all elements of the `matrix` in spiral order. Your code should work on any matrix (a nested list) with $m > 0$ and $n > 0$.
Mark: [10%]

Example 1:



Input: `matrix = [[1,2,3],[4,5,6],[7,8,9]]`
Output: `[1,2,3,6,9,8,7,4,5]`

Example 2:



Input: `matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]`
Output: `[1,2,3,4,8,12,11,10,9,5,6,7]`

Hint: You are allowed to use numpy module. In that case, make sure your function's output is converted to a nested list before returning the output.

Alternatively, you can use the approach below:

1. Initialize the top, right, bottom, and left boundaries as `up`, `right`, `down`, and `left`.
2. Initialize the output array `result`.
3. Traverse the elements in spiral order and add each element to `result`:
   - Traverse from `left` boundary to `right` boundary.
   - Traverse from `up` boundary to `down` boundary.
   - Before we traverse from right to left, we need to make sure that we are not on a row that has already been traversed. If we are not, then we can traverse from `right` to `left`.
   - Similarly, before we traverse from top to bottom, we need to make sure that we are not on a column that has already been traversed. Then we can traverse from `down` to `up`.
   - Remember to move the boundaries by updating `left`, `right`, `up`, and `down` accordingly.
4. Return `result`.

Question 3) You are given four lists `nums1`, `nums2`, `nums3`, and `nums4`, all of them with integer numbers as their items, and all of them with equal length `n`. Write a function `Exercise3(nums1, nums2, nums3, nums4)` that returns the **number of tuples** `(i, j, k, l)` such that:

```
0 <= i, j, k, l < n
nums1[i] + nums2[j] + nums3[k] + nums4[l] == 0
```

Mark: [10%]

Example 1:

Input: `nums1 = [1,2], nums2 = [-2,-1], nums3 = [-1,2], nums4 = [0,2]`
Output: `2`
Explanation:
The two tuples are:
1. (0, 0, 0, 1) -> nums1[0] + nums2[0] + nums3[0] + nums4[1] = 1 + (-2) + (-1) + 2 = 0
2. (1, 1, 0, 0) -> nums1[1] + nums2[1] + nums3[0] + nums4[0] = 2 + (-1) + (-1) + 0 = 0

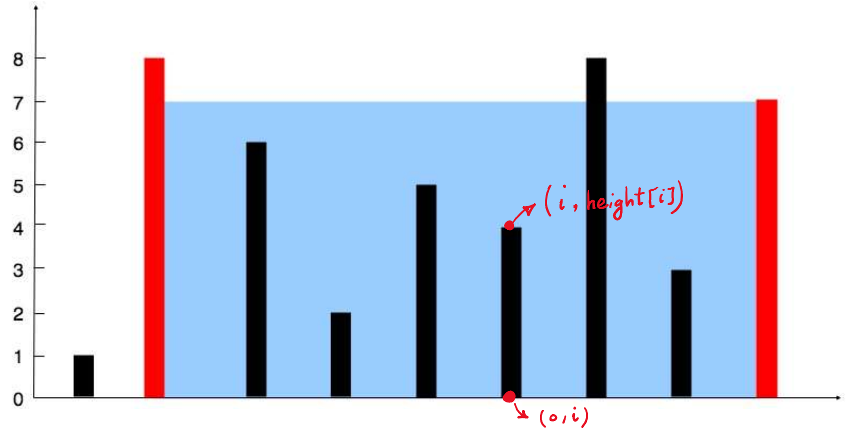
Example 2:

Input: `nums1 = [0], nums2 = [0], nums3 = [0], nums4 = [0]`
Output: `1`

Hint:
An easy and efficient approach would be to use three nested loops. In this case, for each sum `a+b+c`, search for a complementary value `d == -(a+b+c)` in the fourth array. It is recommended that you populate the fourth array into a dictionary.
Note that you need to track the frequency of each element in the fourth array. If an element is repeated multiple times, it will form multiple quadruples. Therefore, we will use dictionary values to store counts.

Question 4) You are given an integer list `height` of length `n`. The integers inside the list are interpreted as `n` vertical lines drawn such that the two endpoints of the `i-th` line (in a cartesian plane) are `(i, 0)` and `(i, height[i])`. This is illustrated below in red.
Write a function `Exercise4(height)` such that it finds two lines that together with the x-axis form a container, such that the container contains the most water.
Return the maximum amount of water a container can store.
Mark: [10%]

Illustration:



Hint (Two Pointer Approach): I recommend you review the two pointer approach in our reading list before approaching this exercise.

Intuitively, the area formed between the lines will always be limited by the height of the shorter line. Furthermore, the farther the lines, the more will be the area obtained.
We take two pointers, one at the beginning and one at the end of the array constituting the length of the lines. Further, we create and keep a variable `maxarea` to store the maximum area obtained till now. At every step, you have to find out the area formed between them, update `maxarea` and move the pointer pointing to the shorter line towards the other end by one step.

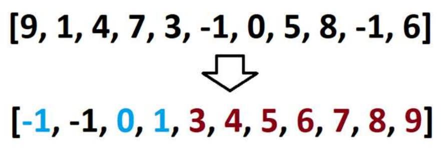
How does this approach work?
First consider the area constituting the exterior most lines. Then, to maximize the area, consider the area between the lines of larger lengths. If you try to move the pointer at the longer line inwards, you won't gain any increase in area, since it is limited by the shorter line. But moving the shorter line's pointer, it could turn out to be beneficial, as per the same argument (despite the reduction in the width). This is done since a relatively longer line obtained by moving the shorter line's pointer might overcome the reduction in area caused by the width reduction.

Question 5) You are given an unsorted list of integers `nums`.
Write a function `Exercise5(nums)` that returns the length of the longest consecutive elements sequence.
Mark: [10%]

Example 1:
Input: `nums = [100,4,200,1,3,2]` **Output:** `4` **Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore, its length is 4.
Example 2:
Input: `nums = [0,3,7,2,5,8,4,6,0,1]` **Output:** `9`

Hint:
Intuitively, if we can iterate over the numbers in ascending order, then it will be easy to find sequences of consecutive numbers. To do so, we can sort the array (You could use built-in method `sort()` to sort the input list if that makes your code easier).

Before we do anything, we check for the base case input of the empty array. The longest sequence in an empty array is, of course, 0, so we can simply return that. For all other cases, we sort `nums` and consider each number after the first (because we need to compare each number to its previous number). If the current number and the previous are equal, then our current sequence is neither extended nor broken, so we simply move on to the next number. If they are unequal, then we must check whether the current number extends the sequence (i.e., `nums[i] == nums[i-1] + 1`). If it does, then we add to our current count and continue. Otherwise, the sequence is broken, so we record our current sequence and reset it to 1 (to include the number that broke the sequence). It is possible that the last element of `nums` is part of the longest sequence, so we return the maximum of the current sequence and the longest one.



In the above image, an example list is shown which has been sorted before the linear scan identifies all consecutive sequences. The longest sequence then is colored in red.

Question 6) You are given a list of integers `nums` containing `n+1` integers where each integer is in the range `[1,n]` inclusive. There is only **one repeated number** in `nums`. Write a function `Exercise6(nums)` that returns this repeated number.
Mark: [10%]

Example 1:

Input: `nums = [1,3,4,2,2]`
Output: `2`

Example 2:

Input: `nums = [3,1,3,4,2]`
Output: `3`

Hint:
In an unsorted array, duplicate elements may be scattered across the array. However, in a sorted array, duplicate numbers will be next to each other. You may use the method `sort()` as well if that makes your code easier.

Question 7) You are given a string `s` and an integer `k`. Write a function `Exercise7(s,k)` that returns the length of the longest substring of `s` that contains at most `k` **distinct** characters.
Mark: [10%]

Example 1:
Input: `s = "eceba", k = 2` **Output:** `3` **Explanation:** The substring is "ece" with length 3.
Example 2:
Input: `s = "aa", k = 1` **Output:** `2` **Explanation:** The substring is "aa" with length 2.

Hint:
To solve this problem in one pass, you need to use sliding window approach with two set pointers `left` and `right` serving as the window boundaries.
The plan is that you set both pointers in the position `0` and then move `right` pointer to the right while the window contains not more than `k` distinct characters. If at some point you got `k + 1` distinct characters, then you may increase `left` pointer to keep not more than `k + 1` distinct characters in the window.

Question 8) You are given a list of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position. Write a function `Exercise8(nums,k)` which returns the max number of the sliding window. Note that `k` is always between 1 and `len(nums)`. Also, `len(nums)` is always greater than or equal to 1. Additionally note that you function should return the result in the form of a list.
Mark: [10%]

Example 1:

Input: `nums = [1,3,-1,-3,5,3,6,7], k = 3`
Output: `[3,3,5,5,6,7]`
Explanation:

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Example 2:

Input: `nums = [1], k = 1`
Output: `[1]`

Question 9) You get two strings `s` and `t` of lengths `m` and `n` respectively. Write a function `Exercise9(s, t)` that returns the **minimum window substring** of `s` such that every character in `t` (**including duplicates**) is included in the window. If there is no such substring, return the empty string `""` (note that there is no space between quotation marks).
Your solution will be tested against inputs whose answer is **unique**.
Mark: [10%]

Example 1:

Input: `s = "ADOBECODEBANC", t = "ABC"`
Output: `"BANC"`
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:

Input: `s = "a", t = "a"`
Output: `"a"`
Explanation: The entire string s is the minimum window.

Example 3:

Input: `s = "a", t = "aa"`
Output: `""`
Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.

Question 10) Design a class with data structure properties that follows the constraints of a Least Recently Used (LRU) cache. Implement the `Cache` class (Your class name must be `Cache(OrderedDict)`) such that:

- `Cache(int capacity)` Initialize the LRU cache with **positive** size `capacity`. Note that in our tests, `capacity` is between 1 and 3000.
- `int get(int key)` returns the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` updates the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

Mark: [10%]

Example:

Input
["Cache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
```
C = Cache(2);
C.put(1, 1); // cache is {1=1}
C.put(2, 2); // cache is {1=1, 2=2}
C.get(1);    // return 1
C.put(3, 3); // cache now is {1=1, 3=3}
C.get(2);    // returns -1 (not found)
C.put(4, 4); // cache now is {4=4, 3=3}
C.get(1);    // return -1 (not found)
C.get(3);    // return 3
C.get(4);    // return 4
```

Hint for the answer with the full mark:
The solution can be written with a dictionary data structure.
You're asked to implement the structure which provides the following operations:

- Get the key / Check if the key exists
- Put the key
- Delete the first added key