Assignment: Portfolio

I. For the problem A answer the below questions

Think of an approach to solve the problem, let us say approach 1 (Brute force is a valid approach to think of)

1. Write the pseudocode / Description of approach 1

Approach one - brute force/recursive approach

```
def helper_1(str1,str2,index1,index2):
    if not index1: return index2
    if not index2: return index1
    if str1[index1-1] == str2[index2-1]:
        return helper_1(str1,str2,index1-1,index2-1)
    res = 1 + min( helper_1(str1, str2,index1-1,index2),
helper_1(str1, str2, index1, index2 -1) )
    return res
def checkPalindrome_1(string, k):
    rev = string[::-1]
    length = len(string)
    return (helper_1(string,rev,length,length) <= k*2)
```

2. Implement your solution

3. What is the time complexity of the approach 1

The time complexity of this approach is given by its worst case, when the string has all unique characters. This gives us 2 recursive calls for each call until all subproblems have been looked at. Thus O(2^n) worst case.

Think of a different approach to solve the problem, let us call this approach 2

4. Write the pseudocode / Description of approach 2

Approach two - dynamic programming

```
   def checkPalindrome_2(string, k):
       rev = string[::-1]
       index1 = index2 = len(rev)
       darr = [[0] * (index1 + 1) for uselessVariable in
range(index1 + 1)]
       for i in range(index1+1):
           for j in range(index2+1):
               if not i or not j:
                   darr[i][j] = 0
               elif (string[i - 1] == rev[j - 1]):
                   darr[i][j] = darr[i-1][j-1]
               else:
                   darr[i][j] = 1 +
min(darr[i-1][j],darr[i][j-1])
```

```
        return darr[index1][index2] <= k * 2
```

This solution is essentially a modified version of the longest common subsequence.

5. Implement your solution

6. What is the time complexity of approach 2

The time complexity here is based of the time to population the array known as darr[] Each iteration of the double for loop fills one cell each time. So the time complexity is Θ(n^2)

7. (Bonus) Does your approach 2 have improved time complexity compared to approach 1, if yes compare the time complexities. The bonus points will be awarded only if your approach 2 is an improvement over approach 1. **The first approach has a bad complexity of 2^n worst case which is horrible. The second approach is using a n^2 average case. This is a much better improvement over the initial approach. The 2^n blows the n^2 out of the water for sufficiently larged sized n.**

8. (Bonus) How did you use the techniques learnt in this course to solve this problem? Name the algorithm techniques/strategies that you have used. (Brute force is not counted as an answer) **So the first approach I used was basic brute force. To solve the problem. After looking at how the data worked and was transferred/ used in this problem. I saw a potential use because of optimal substructure and overlapping subproblems. Thus I decided to look back at some implementations of dynamic programming. I noticed that the longest common subsequence problem could be reduced to the palindrome problem because the palindrome can be verified the same way. So the idea here is to reverse the entered string then check for lcs.**

II. For the problems B and C answer the below questions

Problem b

1. Write a pseudocode / Description to solve the problem

Approach Used - Dynamic programming

```
def patternmatch(string, p):
    n = len(string)
    m = len(p)
    cache = [[False for i in range(m + 1)] for j  in range(n+1)]
    cache[0][0] = True
    for j in range(1, m+1):
        if p[j - 1] == '*':
            cache[0][j] = cache[0][j-1]
    for i in range(1, n+1):
        for j in range(1, m+1):
```

```
            if p[ j - 1 ] == '*':
                cache[i][j] = cache[i][ j - 1] or cache[i - 1][j]
            elif p[ j - 1 ] =='?' or string[ i - 1 ] == p[ j - 1 ]:
                cache[i][j] = cache[i-1][j-1]
            else:
                cache[i][j] = False
    return cache[m][n]
```

This is a modified version of the longest common subsequence problem… Again

2. Implement your solution
3. What is the time complexity of your solution?

The time complexity here is determined by the time it takes to populate the cache array. Thus we get A Θ(n*m) where n is the number of chars in the string and m is the number of chars in the pattern.

4. (Bonus) How did you use the techniques learnt in this course to solve this problem? Name the algorithm techniques/strategies that you have used. (Brute force and Recursion are not counted as an answer)

I just used what I did in the last problem. I took a look at past examples to see if the problem could be reduced to anything. As it turns out it could be reduced to the longest common subsequence problem. Which can be solved in poly nominal time. The differences in this problem are the use of logic and the data stored in the table.

Problem c

1. Write a pseudocode / Description to solve the problem

Approach Used -GreedyAlgorithm

```
def getTesla(M):
    netChange = M[0][0]
    count = 0
    i = j = 0
    rowLen = len(M)-1
    colLen = len(M[0])-1
    while(i < rowLen and j < colLen):
        if(M[i+1][j] < M[i][j+1]):
            j = j + 1
        else:
            i = i + 1
        netChange += M[i][j]
        if(netChange <= 0):
            temp = toOne(netChange)
            netChange+=temp
            count+=temp
```

```
while(i < rowLen):
    i += 1
    netChange += M[i][j]
    if(netChange <= 0):
        temp = toOne(netChange)
        netChange+=temp
        count+=temp

while( j < colLen):
    j += 1
    netChange += M[i][j]
    if(netChange <= 0):
        temp = toOne(netChange)
        netChange+=temp
        count+=temp
return count
```

This algorithm uses the greedy algorithm to traverse this maze. While it traverses the maze making the optimal choice based on what move benefits it the most, it logs the affect each move has on "Mr X". If the move ends up killing mr x, mr x is "revived" by added the minimum amount of health points back to the total to make him have at least one health point for the next move. Every health point added is logged as well. When the function is terminated it returns how points were added to mr x at alive.

2. Implement your solution

3. What is the time complexity of your solution?

The time complexity of this solution is determined by the time it takes to get from mr x to the tesla. Because these spots are fixed we get a Θ(N+M) where n is the length of a column and m is the length of a row.

4. (Bonus) How did you use the techniques learnt in this course to solve this problem? Name the algorithm techniques/strategies that you have used. (Brute force and Recursion are not counted as an answer)

I used the nearest neighbor heuristic to obtain an approximate solution. This stems from the greedy algorithm technique where the program attempts to make the optimal choice.

Problem A:

Name your function for approch1: **checkPalindrome_1(string, k)**

Name your function for approch2: **checkPalindrome_2(string, k)**

File name: **Palindrome.py**

You might be aware of palindrome strings. A word is a palindrome if it reads the same when it is reversed. Example: 'madam', 'refer'

Given a string, find out if the string is a k-palindrome or not. A k-palindrome string becomes a palindrome string when at most k characters are removed from the original string. Return True if the string is k-palindrome, else return False. Return type is Boolean.

Assume that characters in the string will be of uniform case.

Examples:

1. Input: string: "abcdcba"; k =1

   Output: True

   Explanation: When 0 characters are removed the string is palindrome, hence it is 1-palindrome. Optionally if we remove 1 character d from string "abcdcba", the string becomes a 1-palindrome string – "abccba".
2. Input: string: "abcdeba"; k =2

   Output: True

   Explanation: By removing characters 'de' from the string it becomes a 2-palindrome string.

Problem B:

Name your function: **pattermatch(string, p)**

File name: **Patternmatch.py**

Have you heard of pattern matching? Before we dive into the problem, let's see an example. It will be helpful to know some terminology.

- '?' can match any single character
- '*' can match any sequence of characters (including empty sequence)

Let us see an example of pattern matching. Your function takes string and pattern (p). If the pattern matches the string return True otherwise return False. Return type is Boolean.

Example 1: String: "abcde", pattern: "*".

Output: True

The pattern matches the string as * can match any sequence of characters.

Example 2: String: "abcde", pattern: "*a?c*"

Output: True

Let's match this backwards to understand this better. The last '*' matches with sequence "de". '?' matches with "b". Then 'a' matches with "a". We have an extra '*', which can match with an (imaginary) empty sequence before "a".

Example 3: String: “abcde”, pattern: “ad”

Output: False

The pattern does not match with the string

Example 4: String: “abcde”, pattern: “ad?”

Output: False

The patten does not match with the string

Problem C:

Name your function: **getTesla(M)**

File name: **GetTesla.py**

Mr.X is in a maze. Tesla’s latest model car’s keys (K) are in the bottom right corner of the maze. Mr.X is located at the top left corner of the room. The rooms are connected in such a way that Mr.X can move only right or down in a single step.

At each step there is a energy booster (positive integers) or a energy exhausting space (represented by negative integers) or neutral space (represented by 0). As X enters that space, X would either gain health or lose health. The first room and the last room would also impact Mr.X’s energy levels.

To begin with Mr.X starts with some health points. X might lose or gain health points as X enters into different spaces in the maze.

Find the initial health points needed for Mr.X to be able to win the keys (K). Your function should return the health points value. The input maze M is provided as a 2-D array as shown in the example below.

Example:

-1 (X)	-2	2
10	-8	1
-5	-2	-3 (K)

The minimum health points needed are 2. X would follow the path

down->down->right->right

Sample Input: M: [[-1, -2, 2], [10, -8, 1], [-5, -2, -3]]; Output: 2

Debriefing (required!): ----------

Report:

1. Approximately how many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. How deeply do you feel you understand the material it covers (0%–100%)?
4. Any other comments?

Note: ‘Debriefing’ section is intended to help us calibrate the assignments.