Lab 4

1 Exercises

1. Given the names and grades for each student in a class of students, store them in a nested list and print the name(s) of any student(s) having the second lowest grade.
   Note: If there are multiple students with the second lowest grade, order their names alphabetically and print each name on a new line.

2. Given a string, we'll say that the front is the first 3 chars of the string. If the string length is less than 3, the front is whatever is there. Return a new string which is 3 copies of the front.
   Examples: front3('Python') → 'PytPytPyt' or front3('ab') → 'ababab'.

3. Given 2 arrays of ints, a and b, return True if they have the same 3 consecutive ints. Both arrays will be length 3 or more.
   Examples: [1, 10, 9, 10, 8] and [8, 10, 9, 7, 6] → False
   [1, **10, 9, 10**, 8] and [**10, 9, 10**, 7, 6] → True

4. Write a Python program to count the frequency of words in a file. Store them in dictionary, where key represents the word and the value represents the frequency.

5. You are given a list of elements. Identify the unique elements together with how many times they appeared in the list. Print the most common n elements. HINT: you can also use Counter

```
from collections import Counter
```

6. You are given two lists a and b of equal length. Use zip function to print on each line one element from a and one element from b.

```
Example: a=[1,2,3], b=["a", "b", "a"]
1 "a"
2 "b"
3 "a"
```

7. You are given a list of lists. Flatten the list.

8. Implement a class Geometric Form with two methods: getArea and getPerimeter. Using inheritance, implement also Rectangle and Circle with the corresponding functions for area and perimeter.

9. Implement a class for complex numbers a+bi represented as a tuple of (a,b). Implement add, difference, product operations.

```
no1=Complex(3,2)
no2=Complex(5,6)
no1+no2 should return (8, 8)
no1-no2 should return (-2, -4)
np1*no2 should return (3, 28)
```

10. Start from the graph problem from Lab3. Compute the adjacency matrix for the graph.

11. Starting from the given code, implement a SearchProblem for pacman in a grid world. Initially, pacman is in a position (x,y) and wants to get to a given position (xo, y0). The known actions are up, down, left, right.

```
lass SearchProblem:
    def __init__(self, n, x_goal, y_goal):
        ....
    def get_initial_state(self):
        .....

    def is_goal(self, state):
        .....

    def get_successors(self, state):
        """ Only the legal actions from the given state . It
            returns a list of pairs (action, state), where a state
            is (x,y), and action is one of up, down, left, right.
            """
        ......

    def do_path(self, path, state):
        """starting from the given state, execute the path of
            actions and print the final state. the path is a list
            of actions"""
        .......
```

Help pacman achieve its goal by writing a function (outside the SearchProblem class) that chooses **randomly** an action from the legal actions returned by get_succesors method.

12. Let's play with PyGameZero. Read the description at Snake game. Now the snake movew according to the pressed key: up, down, left, right.

    - Change the control of the snake to a random movement: when any of the above keys is pressed, the snake moves randomly to one adjacent position.
    - Change to control of the snake to a more intelligent behavior: it moves to the cell closest to the food.
    - Let's make it a smart one: implement a depth first search algorithm that computes the complete path towards the food and then executes it (at each key pressed, one step of the solution). Once the food is eaten and a new food appears, redo the search step.

13. A robot moves in a plane starting from the original point (0,0). The robot can move toward UP, DOWN, LEFT and RIGHT with a given steps. The trace of robot movement is a sequence of tuples as shown in the following:

    (UP, 5)
    (DOWN, 3)
    (LEFT, 3)
    (RIGHT, 2)

    The numbers after the direction are steps. Write a program to compute the distance from current position after a sequence of movement and original point. If the distance is a float, then just print the nearest integer.

    HINT: To get the he nearest integer use the **round()** function

    Example:

```
Input: [("UP", 5), ("DOWN", 3), ("LEFT", 3), ("RIGHT", 2)]
Output: 2
```

14. You are asked to write a discount system for a beauty saloon, which provides services and sells beauty products. It offers 3 types of memberships: Premium, Gold and Silver. Premium, gold and silver members receive a discount of 20%, 15%, and 10%, respectively, for all services

provided. Customers without membership receive no discount. All members receives a flat 10% discount on products purchased (this might change in future). Your system shall consist of three classes: Customer, Discount and Visit, as shown in the class diagram. It shall compute the total bill if a customer purchases x of products and y of services, for a visit.

HINT: Use the datetime class from the datetime module for the date attribute of the Visit class.

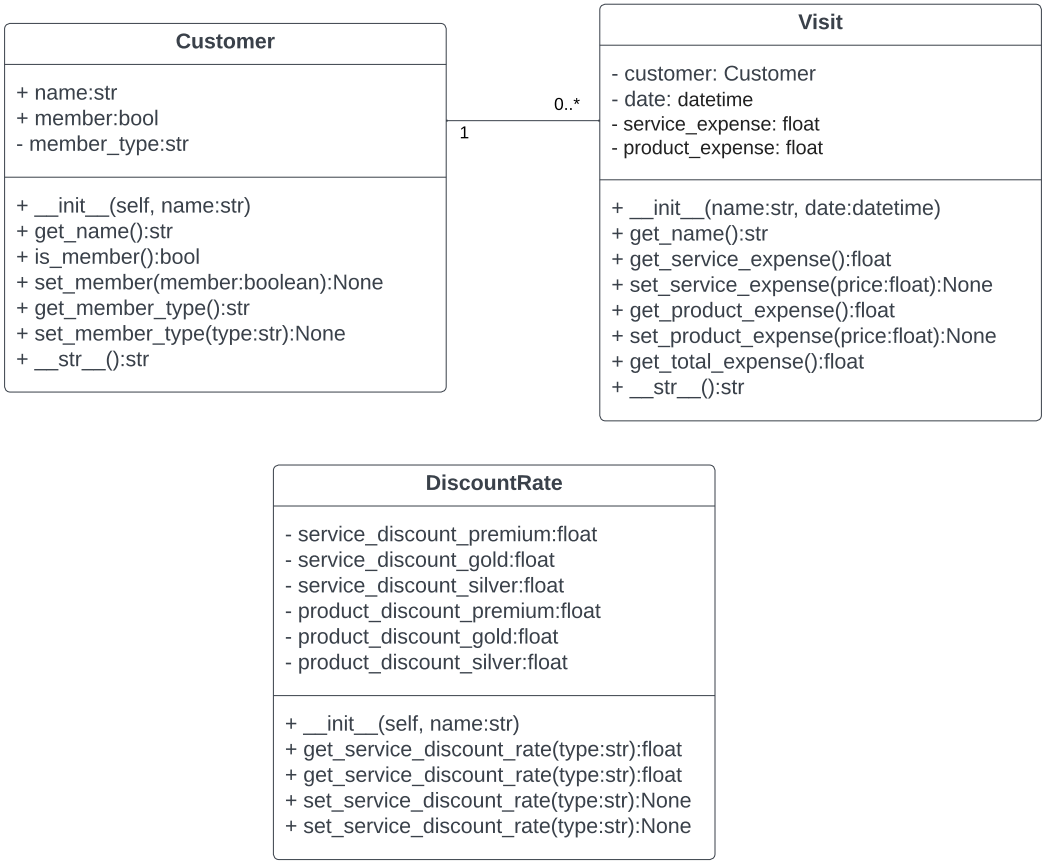


Figure 1: UML Diagram

15. Write a function that takes two inputs n and m and outputs the number of unique paths from the top left corner to bottom right corner of an n x m grid. Constraints: You can only move down or right 1 unit at the time.

HINT: Use Recursion

Example 1:

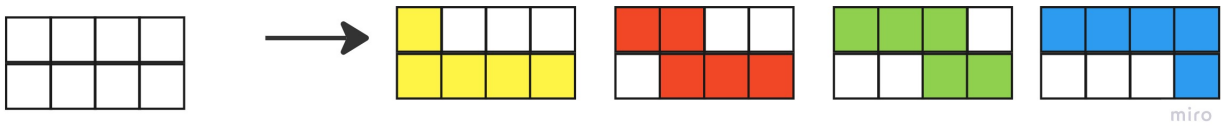
```
Input: m = 1, n = 3
Output: 1
```

Explanation:

Example 2:

```
Input: m = 2, n = 4
Output: 4
```

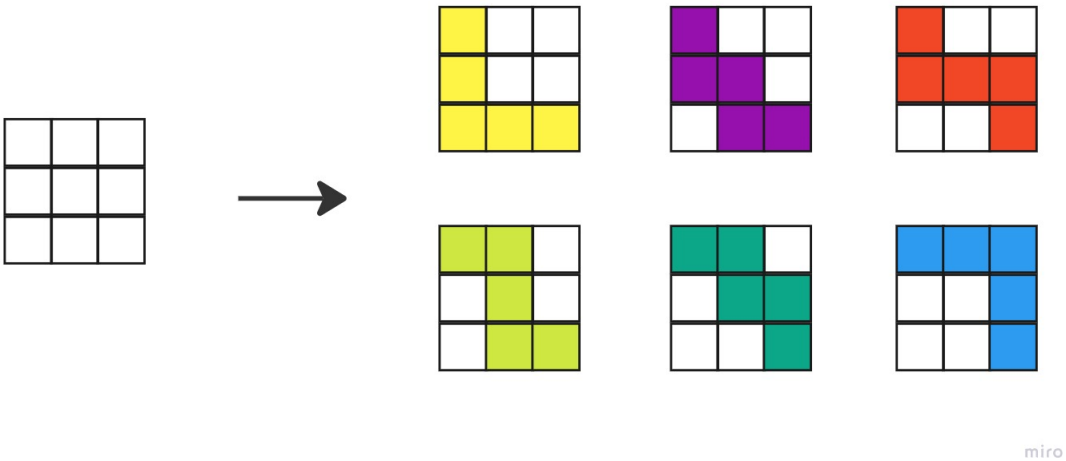
Explanation:



Example 3:

```
Input: m = 3, n = 3
Output: 6
```

Explanation:

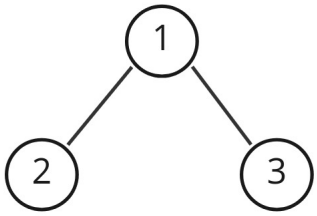


16. Given the roots of two binary trees p and q, write a function to check if they are the same or not. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value. To represent the nodes use the following class:

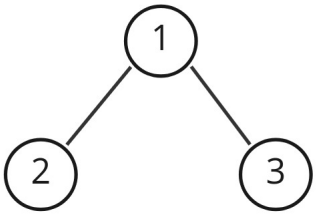
```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

where **self.left** is the root node of the left subtree and **self.right** is the root node of the right subtree. **self.val** is the node value.

Example 1:



Inorder: 2 1 3
pre-order: 1 2 3
post-order: 2 3 1

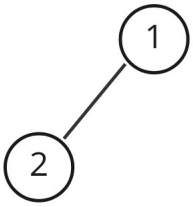


Inorder: 2 1 3
pre-order: 1 2 3
post-order: 2 3 1

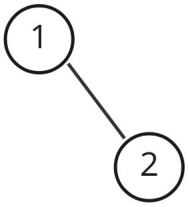
miro

```
Input: p = node(1,node(2),node(3)),q = node(1,node(2),node(3))
Output: True
```

Example 2:



Inorder: 2 1
pre-order: 1 2
post-order: 2 1

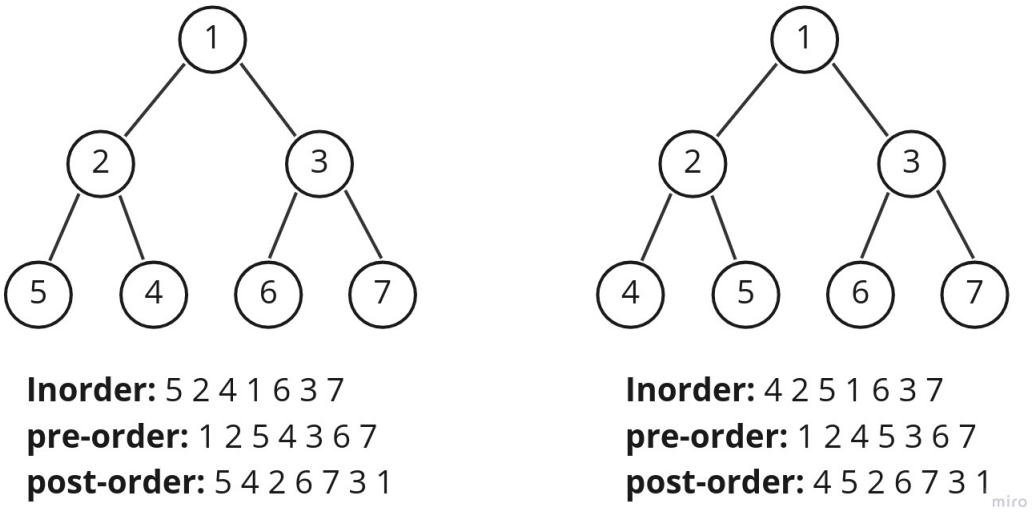


Inorder: 1 2
pre-order: 1 2
post-order: 2 1

miro

```
Input: p = node(1,node(2),null),q = node(1,null,node(2))
Output: False
```

Example 3:




```
Input: p = node(1,node(2),node(3)),q = node(1,node(2),node(3))
Output: False
```

17. **Vectors**

- Read the file *vectors.txt* which has the following structure. N represents the number of vectors in the file and x, y, z represent the coordinates of each vector. Store these vectors into a dictionary.

```
n
x1 y1 z1
x2 y2 z2
x3 y3 z3
...
```

- Compute the mean for each coordinate e.g. μ_x, μ_y, μ_z

$$\mu_x = \frac{\sum_{i=1}^{n} x_i}{n}$$

- Compute the standard deviation for each coordinate e.g. $\sigma_x, \sigma_y, \sigma_z$

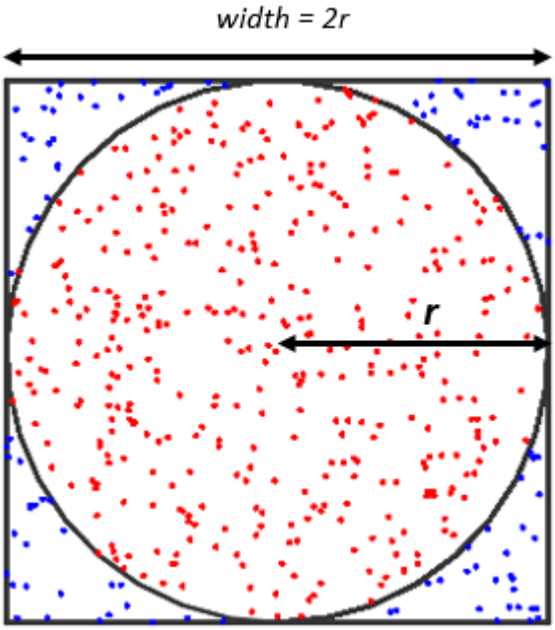
$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu_x)^2}{n}}$$

- Check if your operations do the same thing as **numpy.mean()** and **numpy.std()** from numpy package. If you don't have the package already, use this command to install it:

```
pip install numpy
```

- Create a class for vector manipulation. You can be creative with the way you organize your data. In this class you will have to implement some methods for vector operations: addition, subtraction, dot product and cross product. (You can also check the correctness of these operations using the numpy methods.)

18. **Monte Carlo PI**



Compute PI number using Monte Carlo method. The Monte Carlo method relies on repeated random sampling to obtain numerical results. In this exercise we would like to obtain the transcendental number PI. For this we have to sample points with 2D coordinates (x, y) randomly. Then we will consider two sets of points: those that are just inside a circle of radius R and those which are inside a square of length $2 * R$ (*all of them*). Now, where does PI come into the scene, you may ask. As you know the area of a circle in $pi * r^2$ and the area of a square is r^2. For the example in the image, if we divide the two areas we get $pi/4$. But, since we can't compute the area of a circle without PI itself we approximate that area by the number of random points lying on that surface. Hence, the solution is the following:

$$\pi = 4 * \frac{|circle\_points|}{|square\_points|}$$

The precision of PI is dependant on the number of randomly sampled points.

19. **Golden Ration**

Compute Golden Ration using Fibonacci numbers. The Golden Ratio represents an important number in mathematics. It has been discovered that it could be computed by dividing big consecutive Fibonacci numbers. So, for this problem you have to come with an efficient algorithm that computes the Fibonacci series. Then select the last two numbers generated by yourself and divide them (last over the first). The precision for the Golden Ratio is dependant on the number of digits that the Fibonacci elements have.

$$\phi = 1.61803398875$$