Week-5, Practice, Programming

Problem-1

Question

Solve the following system of linear equations:

$$a_1 x + b_1 y + c_1 = 0$$
$$a_2 x + b_2 y + c_2 = 0$$

Assumptions

- All coefficients are non-zero integers.
- The system has a unique solution.
- The solution will always be a pair of integers.

Task

The coefficients will be given as a list of integers:

- `eq1`: $[a_1, b_1, c_1]$
- `eq2`: $[a_2, b_2, c_2]$

Write a function named `solve` that accepts these two lists as input and returns the solution as output. Do not modify the name of the function.

```
def solve(eq1, eq2):
    '''Return solution (x, y) as output'''
    pass
```

Note

- You don't need to accept the input from the user or print the output to the console. This will be processed internally.
- You only need to fill the details in the body of the function.

Test Cases

Type	Input	Output
Public	`1 -2 6` `3 5 -15`	`0 3`
Public	`10 -4 -58` `6 4 10`	`3 -7`
Private	`2 -1 -3` `3 2 -1`	`1 -1`
Private	`-3 2 34` `5 -3 -56`	`10 -2`

Answer

```
def solve(eq1, eq2):
    '''Return (x, y) as output'''
    # The string in triple quotes above
    # is called a doc string.
    a1, b1, c1 = eq1    # coefficients for eq1
    a2, b2, c2 = eq2    # coefficients for eq2
    # What we have done in the code given above
    # is called sequence unpacking.
    # We have unpacked the list eq1 into the
    # three coefficients. Ditto for eq2.
    # Since there are three elements in each list,
    # those three are going to be assigned to
    # the three variables ai, bi, ci.

    # Simple substitution by eliminating
    # the variables x and y
    x = (b1 * c2 - b2 * c1) / (a1 * b2 - a2 * b1)
    y = (c1 * a2 - c2 * a1) / (a1 * b2 - a2 * b1)

    # The assumptions are useful here.
    # We can convert x and y to int.
    # We won't be losing information that way.
    return int(x), int(y)
```

Suffix Code Block

```
# This method of accepting input is called list-comprehension
eq1 = [int(word) for word in input().split()]
# List comprehension will be covered in upcoming weeks
eq2 = [int(word) for word in input().split()]
x, y = solve(eq1, eq2)
print(x, y)
```

Problem-2

Question

Find the transpose of a given matrix.

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix}$$

Assumptions

- All cells in the matrix are integers.
- The dimension of the matrix is $m \times n$, where $m, n \geq 1$. The first line in the input will have the dimension as space-separated integers.
- The next m lines in the input will be a sequence of n space-separated integers.
- The output will be the transpose of this matrix. Each row of the matrix is given as a sequence of space-separated integers.

Task

Write a function `transpose` that accepts a matrix as input and returns its transpose. You can assume that the matrix is a nested list.

```
def transpose(mat):
    '''Returns transpose of mat'''
    pass
```

Note

- You don't need to accept the input from the user or print the output to the console. This will be processed internally.
- You only need to fill the details in the body of the function.

Test Cases

Type	Input	Output

Public	`3 2`
	`1 2`
	`3 4`
	`5 6`
		`1 3 5`
		`2 4 6`

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

Private	`1 5`
	`1 2 3 4 5`
		`1`
		`2`
		`3`
		`4`
		`5`

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

Answer

```python
def transpose(mat):
    '''Return transpose of mat'''
    trans = [ ]
    # mat is of dimensions m x n
    # len(mat) is the number of rows
    # mat[0] is the first row
    # len(mat[0]) is the number of columns
    m, n = len(mat), len(mat[0])
    # While printing a matrix, we do the following:
    # We go from left-> right, then top->bottom (row-first)
    # To get the transpose, we do the following
    # We go from top->bottom, then left->right (column-first)
    for j in range(n):
        trans.append([ ])
        for i in range(m):
            trans[j].append(mat[i][j])
    return trans
```

Suffix Code Block

```python
dims = input().split()
m, n = int(dims[0]), int(dims[1])
mat = [ ]
for i in range(m):
    mat.append([ ])
```

```
    for num in input().split():
        mat[i].append(int(num))

trans = transpose(mat)
for i in range(n):
    for j in range(m):
        if j != m - 1:
            print(trans[i][j], end = ' ')
        else:
            print(trans[i][j])
```

Problem-3

Question

In large-scale programming projects, organizing files and folders becomes crucial. Consider the following folder structure. The path to a file is expressed in the following manner:

```
1 /home
2 /home/mark
3 /home/mark/facebook
4 /home/mark/facebook/src
5 /home/mark/facebook/src/newsfeed.py
```

- Line-1: The `/home` folder is like a huge palace. Users have their own rooms in it. This folder is at level-1.
- Line-2: In the `home` folder, we have a user named `mark`. `/home/mark` is his room. This folder is at level-2.
- Line-3: Mark is working on a project called `facebook`, naturally within his room. This folder is at level-3.
- Line-4: Within this project, he maintains a folder named `src` to store all his `Python` files. This folder is at level-4.
- Line-5: The path points to `newsfeed.py`, one of the files in the folder. This file is at level-5.

The level of a file or folder quantifies its depth within the folder structure.

Assumptions

- There are only two types of entities: files and folders.
- Files could end with one of the following extensions:
  - `.py` — code
  - `.cpp` — code
  - `.jpg` — image
  - `.png` — image
- Unlike the chicken-egg problem, only files can reside in folders. Folders cannot be present in files.
- Folder names will always be alphanumeric characters.

Task

Given a path as input, your task is to write the following functions:

- `is_folder(path)`: this accepts a path as input and returns `True` if the path points to a folder, `False` otherwise
- `is_file(path)`: this accepts a path as input and returns `True` if the path points to a file, `False` otherwise
- `is_code(path)`: this accepts a path as input and returns `True` if the path points to a code file and `False` otherwise
- `is_image(path)`: this accepts a path as input and returns `True` if the path points to an image file and `False` otherwise
- `level(path)`: this accepts a path as input and returns the level at which it is found.

Do not print anything to the console. Your task is to write these five functions. The input will be a `path`. The output will be the outcome of the following function calls in this exact sequence:

```
is_folder(path)
is_file(path)
is_code(path)
is_image(path)
level(path)
```

Test Cases

Type	Input	Output
Public	`/home/mark/facebook/src/newsfeed.py`	False True True False 5
Public	`/home/guido/microsoft/secret/mystery/src`	True False False False 6
Public	`/home/einstein/relativity.jpg`	False True False True 3
Private	`/home`	True False False False 1
Private	`/home/numpy`	True False False False 2
Private	`/home/project/something/jpg`	True False False False 4
Private	`/home/random/image.png`	False True False True 3
Private	`/home/d1/d2/d3/d4/d5/file1.cpp`	False True True False 7

Answer

```python
# Look for .py or .cpp extension
def is_code(path):
    if path[-3:] == '.py' or path[-4:] == '.cpp':
        return True
    return False

# Look for .jpg or .png extension
def is_image(path):
    if path[-4:] == '.jpg' or path[-4:] == '.png':
        return True
    return False

# Check if it is a code or an image
def is_file(path):
    return is_code(path) or is_image(path)

# A folder is the negation of a file
def is_folder(path):
    return not is_file(path)

# Split the string based on / and find the length of the resulting list
def level(path):
    return len(path.strip('/').split('/'))
```

Suffix Code Block

```
path = input()
print(is_folder(path))
print(is_file(path))
print(is_code(path))
print(is_image(path))
print(level(path))
```

Problem 4

Question

The Pearson correlation coefficient, `r(x, y)` is a measure of association between two continuous variables `x` and `y`. The correlation `r(x, y)` is given defined in the mathematical form as below.

$$r(x,y) = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}} = \frac{\Sigma_{i=1}^{n}(x_i y_i) - ((\Sigma_{i=1}^{n} x_i)(\Sigma_{i=1}^{n} y_i)/n)}{\sqrt{(\Sigma_{i=1}^{n} x_i^2 - (\Sigma_{i=1}^{n} x_i)^2/n)} * \sqrt{(\Sigma_{i=1}^{n} y_i^2 - (\Sigma_{i=1}^{n} y_i)^2/n)}}$$

Symbol	Meaning
n	number of data points (or count of numbers)
x_i	`i-th` value of the variable x, $i = 1, 2, 3, 4, \ldots, n$
y_i	`i-th` value of the variable y, $i = 1, 2, 3, 4, \ldots, n$
$\bar{x}$	mean or average value of the variable x
$\bar{y}$	mean or average value of the variable y

Implement the body of the function named `pearson_correlation`. The input is two lists of `float` values: `x` and `y`. Each floating point number is limited to `2` places after decimal.

- The function should return the correlation coefficient as a `float` value rounded to `1` decimal place.
- The function should return `0.0`, when the lengths of the two input variables are unequal.

```
def pearson_correlation(x, y):
    # Write function body
    #
    #
    #
```

Test Cases

Public

Input	Output
`1 2 3 4 5 6 7 8 9` `1 2 3 2 3 4 3 4 5`	`0.9`
`1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0` `-1.0 -2.0 -3.0 -4.0 -5.0 -1.0 -2.0 -3.0 -4.0 -5.0`	`-0.5`

Private

Input	Output
`10.0 8.0 13.0 9.0 11.0 14.0 6.0 4.0 12.0 7.0 5.0` `8.04 6.95 7.58 8.81 8.33 9.96 7.24 4.26 10.84 4.82 5.68`	`0.8`
`10.0 8.0 13.0 9.0 11.0 14.0 6.0 4.0 12.0 7.0 5.0` `9.14 8.14 8.74 8.77 9.26 8.10 6.13 3.10 9.13 7.26 4.74`	`0.8`
`10.0 8.0 13.0 9.0 11.0 14.0 6.0 4.0 12.0 7.0 5.0` `7.46 6.77 12.74 7.11 7.81 8.84 6.08 5.39 8.15 6.42 5.73`	`0.8`
`8.0 8.0 8.0 8.0 8.0 8.0 8.0 19.0 8.0 8.0 8.0` `6.58 5.76 7.71 8.84 8.47 7.04 5.25 12.50 5.56 7.91 6.89`	`0.8`
`1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0` `10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0`	`-1.0`
`1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0` `1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0`	`1.0`
`0 1 2 3 4 5 5 4 3 2 1 0` `5 4 3 2 1 0 0 1 2 3 4 5`	`-1.0`
`1.1 2.2 3.3 4.4 5.5 6.6 -7.7 -8.8 -9.9` `-1 -2 -3 -2 -3 -4 3 4 5`	`-1.0`
`1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9` `-1.1 -2.2 -3.3 -4.4 -5.5 -6.6 -7.7 -8.8 -9.9`	`-1.0`

Answer

```python
def pearson_correlation(x, y):
    # convert input (a string of float values separated by spaces) into a list of strings
    # verify the length
    if len(x) != len(y):
        return 0.0
    n = len(x)
    sum_x, sum_y = 0.0, 0.0
    for i in range(n):
        sum_x, sum_y = sum_x + x[i], sum_y + y[i]
    # calculate mean for x and y
    mean_x, mean_y = sum_x / len(x), sum_y / len(y)
    # calculate numerator and denominator
    numerator, denominator_x, denominator_y = 0.0, 0.0, 0.0
    for i in range(n):
        numerator += (x[i] - mean_x) * (y[i] - mean_y)
        denominator_x += (x[i] - mean_x) ** 2
        denominator_y += (y[i] - mean_y) ** 2
    # get pearson correlation coefficient
    denominator =  (denominator_x ** 0.5) * (denominator_y ** 0.5)
    if denominator - 0.0 > 0.00001:
        pearson_corr = numerator / denominator
```

```
    return(round(pearson_corr, 1))
```

Suffix Code Block

```
x, y = input().split(), input().split()
for i in range(len(x)):
    x[i], y[i] = float(x[i]), float(y[i])
print(pearson_correlation(x, y))
```

Alternate Answers

```
def pearson_correlation(x, y):
    # convert input (a string of float values separated by spaces) into a list of strings
    x, y = x.split(), y.split()
    # verify the length
    if len(x) != len(y):
        return 0.0
    n = len(x)
    sum_x, sum_y = 0.0, 0.0
    # change string element to float values in the list x and y
    for i in range(n):
        x[i], y[i] = float(x[i]), float(y[i])
        sum_x, sum_y = sum_x + x[i], sum_y + y[i]

    # calculate numerator and denominator
    sum_x_sq, sum_y_sq, sum_xy = 0.0, 0.0, 0.0
    for xi in x:
        sum_x_sq += xi ** 2
    for yi in y:
        sum_y_sq += yi ** 2
    for i in range(n):
        sum_xy += x[i] *  y[i]
    numerator = sum_xy - (sum_x * sum_y) / n
    denominator = ( (sum_x_sq - sum_x ** 2 / n) * (sum_y_sq - sum_y ** 2 / n) ) ** 0.5
    # get pearson correlation coefficient
    if denominator - 0.0 > 0.00001:
        pearson_corr = numerator / denominator
    return(round(pearson_corr, 1))
```

Verification using library function.

```
from scipy.stats import spearmanr
import numpy
def pearson_correlation(x, y):
    # convert input (a string of float values separated by spaces) into a list of strings
    x, y = x.split(), y.split()
    # verify the length
    if len(x) != len(y):
        return 0.0
    n = len(x)
    sum_x, sum_y = 0.0, 0.0
    # change string element to float values in the list x and y
```

```
    for i in range(n):
        x[i], y[i] = float(x[i]), float(y[i])
    # get pearson correlation coefficient
    # return round(spearmanr(x, y)[0], 1)
    return(round(numpy.corrcoef(x, y)[0][1], 1))
```