

# Rajalakshmi Engineering College

Name: UMAR FAROOK

Email: 240701573@rajalakshmi.edu.in

Roll no: 240701573

Phone: 9791730398

Branch: REC

Department: I CSE FF

Batch: 2028

Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23221\_Python Programming

### REC\_Python\_Week 7\_MCQ

Attempt : 2

Total Mark : 20

Marks Obtained : 19

#### Section 1 : MCQ

1. What does NumPy stand for?

**Answer**

Numerical Python

**Status : Correct**

**Marks : 1/1**

2. Which NumPy function is used to find the indices of the maximum and minimum values in an array?

**Answer**

argmax() and argmin()

**Status : Correct**

**Marks : 1/1**

3. What is the output of the following NumPy code?

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
r = arr[2:4]
print(r)
```

**Answer**

[3 4]

**Status :** Correct

**Marks :** 1/1

4. In the DataFrame created in the code, what is the index for the row containing the data for 'Jack'?

```
import pandas as pd

data = {'Name': ['Tom', 'Jack', 'nick', 'juli'],
        'marks': [99, 98, 95, 90]}

df = pd.DataFrame(data, index=['rank1',
                               'rank2',
                               'rank3',
                               'rank4'])

print(df)
```

**Answer**

rank2

**Status :** Correct

**Marks :** 1/1

5. Which NumPy function is used to calculate the standard deviation of an array?

**Answer**

numpy.std()

**Status :** Correct

**Marks :** 1/1

6. What is the primary data structure used in NumPy for numerical computations?

**Answer**

Array

**Status :** Correct

**Marks :** 1/1

7. What is the output of the following NumPy code snippet?

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
r = arr[arr > 2]
print(r)
```

**Answer**

[3 4 5]

**Status :** Correct

**Marks :** 1/1

8. Which function is used to create a Pandas DataFrame?

**Answer**

pd.DataFrame()

**Status :** Correct

**Marks :** 1/1

9. The important data structure of pandas is/are \_\_\_\_.

**Answer**

Both Series and Data Frame

**Status :** Correct

**Marks :** 1/1

10. In NumPy, how do you access the first element of a one-dimensional array arr?

**Answer**

`arr[0]`

**Status : Correct**

**Marks : 1/1**

11. What will be the output of the following code?

```
import pandas as pnd
pnd.Series([1,2], index= ['a','b','c'])
```

**Answer**

Value Error

**Status : Correct**

**Marks : 1/1**

12. What is the purpose of the following NumPy code snippet?

```
import numpy as np
arr = np.zeros((3, 4))
print(arr)
```

**Answer**

Displays a 3x4 matrix filled with zeros

**Status : Correct**

**Marks : 1/1**

13. Which of the following is a valid way to import NumPy in Python?

**Answer**

```
import numpy as np
```

**Status : Correct**

**Marks : 1/1**

14. What does the `np.arange(10)` function in NumPy do?

**Answer**

Creates an array with values from 1 to 9

Status : Correct

Marks : 1/1

15. What is the result of the following NumPy operation?

```
import numpy as np
arr = np.array([1, 2, 3])
r = arr + 5
print(r)
```

Answer

[6 7 8]

Status : Correct

Marks : 1/1

16. What will be the output of the following code snippet?

```
import numpy as np
arr = np.array([1, 2, 3])
result = np.concatenate((arr, arr))
print(result)
```

Answer

[1 2 3 1 2 3]

Status : Correct

Marks : 1/1

17. Minimum number of argument we require to pass in pandas series ?

Answer

1

Status : Correct

Marks : 1/1

18. Which NumPy function is used to create an identity matrix?

Answer

Status : Skipped

Marks : 0/1

19. What is the output of the following code?

```
import numpy as np  
a = np.arange(10)  
print(a[2:5])
```

**Answer**

[2, 3, 4]

**Status :** Correct

**Marks :** 1/1

20. What is the primary purpose of Pandas DataFrame?

**Answer**

To store data in tabular form for analysis and manipulation

**Status :** Correct

**Marks :** 1/1

# Rajalakshmi Engineering College

Name: UMAR FAROOK  
Email: 240701573@rajalakshmi.edu.in  
Roll no: 240701573  
Phone: 9791730398  
Branch: REC  
Department: I CSE FF  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23221\_Python Programming

### REC\_Python\_Week 7\_COD

Attempt : 1  
Total Mark : 50  
Marks Obtained : 50

### Section 1 : Coding

#### 1. Problem Statement

Alex is a data scientist analyzing the relationship between two financial indicators over time. He has collected two time series datasets representing daily values of these indicators over several months. Alex wants to understand how these two indicators correlate at different time lags to identify possible leading or lagging behaviors.

Your task is to help Alex compute the cross-correlation of these two time series using numpy, so he can analyze the similarity between the two signals at various time shifts.

#### ***Input Format***

The first line of input consists of space-separated float values representing the first time series, array1.

The second line of input consists of space-separated float values representing the second time series, array2.

### **Output Format**

The first line of output prints: "Cross-correlation of the two time series:"

The second line of output prints: the 1D numpy array cross\_corr representing the cross-correlation of array1 and array2 across different lags.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 1.0 2.0 3.0  
4.0 5.0 6.0

Output: Cross-correlation of the two time series:  
[ 6. 17. 32. 23. 12.]

### **Answer**

```
import numpy as np

a1 = np.array(list(map(float,input().split())))
a2 = np.array(list(map(float,input().split())))

print("Cross-correlation of the two time series:")
c = np.correlate(a1,a2,mode = 'full')

print(c)
```

**Status :** Correct

**Marks :** 10/10

## **2. Problem Statement**

Sita is analyzing her company's daily sales data to find all sales values that are multiples of 5 and exceed 100. She wants to filter these specific sales values from the list.



Help her to implement the task using the numpy package.

Formula:

To filter sales values:

Select all values  $s$  from sales such that  $(s \% 5 == 0)$  and  $(s > 100)$

### ***Input Format***

The first line of input consists of an integer value,  $n$ , representing the number of sales entries.

The second line of input consists of  $n$  floating-point values, sales, separated by spaces, representing daily sales figures.

### ***Output Format***

The output prints: filtered\_sales

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 5

50.0 100.0 105.0 150.0 99.0

Output: [105. 150.]

### ***Answer***

```
import numpy as np
n = int(input())
a = np.array(list(map(float,input().split()))))
b = a[(a % 5 == 0) & (a > 100)]
print(b)
```

**Status :** Correct

**Marks :** 10/10

## **3. Problem Statement**

A company tracks the monthly sales data of various products. You are given a table where each row represents a product and each column represents its monthly sales in sequential months.

Your task is to compute the cumulative monthly sales for each product using numpy, where the cumulative sales for a month is the total sales from month 1 up to that month.

### ***Input Format***

The first line of input consists of two integer values, products and months, separated by a space.

Each of the next products lines consists of months integer values representing the monthly sales data of a product.

### ***Output Format***

The first line of output prints: "Cumulative Monthly Sales:"

The second line of output prints: the 2D numpy array cumulative\_array that contains the cumulative sales data for each product.

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 2 4  
10 20 30 40  
5 15 25 35

Output: Cumulative Monthly Sales:  
[[ 10 30 60 100]  
[ 5 20 45 80]]

### ***Answer***

```
import numpy as np
```

```
products,months = map(int,input().split())
```

```
a = [list(map(int,input().split())) for _ in range(products)]
```

```
b = np.array(a)
c = np.cumsum(b, axis = 1)

print("Cumulative Monthly Sales:")
print(c)
```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Rekha works in hospital data management and receives patient records with missing or incomplete data. She needs to clean the records by performing the following tasks:

Calculate the mean of the available Age values. Replace any missing (NaN) values in the Age column with this mean age. Remove any rows where the Diagnosis value is missing (NaN). Reset the DataFrame index after removing these rows.

Implement this data cleaning task using the pandas package.

##### **Input Format**

The first line of input contains an integer  $n$  representing the number of patient records.

The second line contains the CSV header — comma-separated column names (e.g., "Name, Age, Diagnosis, Gender").

The next  $n$  lines each contain one patient record in comma-separated format.

##### **Output Format**

The first line of output is the text:

Cleaned Hospital Records:

The next lines print the cleaned pandas DataFrame (as produced by `print(cleaned_df)`).

This will include the updated values of the Age column (with missing ages filled by the mean age), and any rows with missing Diagnosis removed.

The DataFrame will be displayed using the default pandas print() representation.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 5

PatientID,Name,Age,Diagnosis

1,John Doe,45,Flu

2,Jane Smith,,Cold

3,Bob Lee,50,

4,Alice Green,38,Fever

5,Tom Brown,,Infection

Output: Cleaned Hospital Records:

	PatientID	Name	Age	Diagnosis
0	1	John Doe	45.000000	Flu
1	2	Jane Smith	44.333333	Cold
2	4	Alice Green	38.000000	Fever
3	5	Tom Brown	44.333333	Infection

### **Answer**

```
import pandas as pd
```

```
import sys
```

```
import numpy as np
```

```
n = int(input())
```

```
header = input().strip().split(',')
```

```
data = []
```

```
for _ in range(n):
```

```
    line = input().strip().split(',')
```

```
    data.append(line)
```

```
df = pd.DataFrame(data, columns=header)
```

```
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
```

```
mean_age = df['Age'].mean()
```

```
df['Age'] = df['Age'].fillna(mean_age)
```

```
df['Diagnosis'].replace("", np.nan, inplace=True)
```

```
df = df.dropna(subset=['Diagnosis'])
```

```
df.reset_index(drop=True, inplace=True)
print("Cleaned Hospital Records:")
print(df)
```

**Status :** Correct

**Marks : 10/10**

## 5. Problem Statement

Sita works as a sales analyst and needs to analyze monthly sales data for different cities. She receives lists of cities, months, and corresponding sales values and wants to create a pandas DataFrame using a MultiIndex of cities and months.

Help her to implement this task and calculate total sales for each city.

### **Input Format**

The first line of input consists of an integer value,  $n$ , representing the number of records.

The second line of input consists of  $n$  space-separated city names.

The third line of input consists of  $n$  space-separated month names.

The fourth line of input consists of  $n$  space-separated float values representing sales for each city-month combination.

### **Output Format**

The first line of output prints: "Monthly Sales Data with MultiIndex:"

The next lines print the DataFrame with MultiIndex (City, Month) and their corresponding sales values.

The following line prints: "\nTotal Sales Per City:"

The final lines print the total sales per city, computed by grouping the sales data on city names.

Refer to the sample output for the formatting specifications.

**Sample Test Case**

Input: 4

NYC NYC LA LA

Jan Feb Jan Feb

100 200 300 400

Output: Monthly Sales Data with MultiIndex:

Sales

City Month

NYC Jan 100.0

Feb 200.0

LA Jan 300.0

Feb 400.0

Total Sales Per City:

Sales

City

LA 700.0

NYC 300.0

**Answer**

```
import pandas as pd
```

```
n = int(input())
```

```
cities = input().split()
```

```
months = input().split()
```

```
sales = list(map(float, input().split()))
```

```
index = pd.MultiIndex.from_arrays([cities, months], names=["City", "Month"])
```

```
df = pd.DataFrame({"Sales": sales}, index=index)
```

```
print("Monthly Sales Data with MultiIndex:")
```

```
print(df)
```

```
print("\nTotal Sales Per City:")
```

```
print(df.groupby(level="City").sum())
```

**Status : Correct**

**Marks : 10/10**

# Rajalakshmi Engineering College

Name: UMAR FAROOK

Email: 240701573@rajalakshmi.edu.in

Roll no: 240701573

Phone: 9791730398

Branch: REC

Department: I CSE FF

Batch: 2028

Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23221\_Python Programming

### REC\_Python\_Week 7\_PAH

Attempt : 1

Total Mark : 50

Marks Obtained : 49

### Section 1 : Coding

#### 1. Problem Statement

Arjun manages a busy customer service center and wants to analyze the distribution of customer wait times to improve service efficiency. He decides to group the wait times into intervals of 5 minutes each and count how many customers fall into each interval bucket.

Help him implement this bucketing and counting task using NumPy.

Bucketing Logic:

Divide the wait times into intervals (buckets) of size 5 minutes, e.g.:

[0-5), [5-10), [10-15), ...

Use NumPy's digitize function to determine which bucket each wait time falls into.

Count the number of wait times in each bucket and generate bucket labels.

### ***Input Format***

The first line contains an integer  $n$ , the number of customer wait times recorded.

The second line contains  $n$  space-separated floating-point numbers representing the wait times (in minutes).

### ***Output Format***

The first line of output is the text:

Wait Time Buckets and Counts:

Each subsequent line prints the bucket range and the number of wait times in that bucket, formatted as:

<bucket\_range>: <count>

where <bucket\_range> is the lower and upper bound of the bucket (inclusive lower bound, exclusive upper bound), for example:

0-5: 3

5-10: 2

10-15: 1

The output uses the default string formatting of Python's `print()` function (no extra spaces, no special formatting beyond the specified lines).

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 10

2.0 3.0 7.0 8.0 12.0 14.0 18.0 19.0 21.0 25.0



Output: Wait Time Buckets and Counts:

0-5: 2

5-10: 2

10-15: 2

15-20: 2

20-25: 1

**Answer**

```
import numpy as np
```

```
import math
```

```
n = int(input())
```

```
wait_times = list(map(float, input().split()))
```

```
max_time = max(wait_times)
```

```
bucket_size = 5
```

```
upper_limit = math.ceil(max_time / bucket_size) * bucket_size
```

```
bins = list(range(0, upper_limit + bucket_size, bucket_size))
```

```
indices = np.digitize(wait_times, bins, right=False)
```

```
counts = [0] * (len(bins) - 1)
```

```
for idx in indices:
```

```
    if idx <= len(counts):
```

```
        counts[idx - 1] += 1
```

```
print("Wait Time Buckets and Counts:")
```

```
for i in range(len(counts)):
```

```
    print(f"{bins[i]}-{bins[i+1]}: {counts[i]}")
```

**Status : Correct**

**Marks : 10/10**

## 2. Problem Statement

A company conducted a customer satisfaction survey where each respondent provides their RespondentID and an optional textual Feedback. Sometimes, respondents submit their ID without any feedback or with empty feedback.

Your task is to process the survey responses using pandas to replace any missing or empty feedback with the phrase "No Response". Finally, print the cleaned survey responses exactly as shown in the sample output.

### ***Input Format***

The first line contains an integer  $n$ , the number of survey responses.

Each of the next  $n$  lines contains:

A RespondentID (a single alphanumeric string without spaces),

Followed optionally by a Feedback string, which may be empty or missing.

If no feedback is provided after the RespondentID, treat it as missing.

### ***Output Format***

Print the line:

Survey Responses with Missing Feedback Filled:

Then print the cleaned survey data as a table with two columns: RespondentID and Feedback.

The table should have the headers exactly as:

RespondentID Feedback

Print each respondent's data on a new line, aligned to match the output produced by `pandas.DataFrame.to_string(index=False)`.

For any missing or empty feedback, print "No Response" in the Feedback column.

Maintain the spacing and alignment exactly as shown in the sample outputs.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 4

101 Great service

102

103 Loved it

104

Output: Survey Responses with Missing Feedback Filled:

RespondentID	Feedback
--------------	----------

101	Great service
-----	---------------

102	No Response
-----	-------------

103	Loved it
-----	----------

104	No Response
-----	-------------

### **Answer**

```
import pandas as pd
n = int(input())
respondents = []
feedbacks = []
for _ in range(n):
    line = input().strip()
    parts = line.split(' ', 1)
    respondent_id = parts[0]
    feedback = parts[1].strip() if len(parts) > 1 else "
    if not feedback:
        feedback = 'No Response'
    respondents.append(respondent_id)
    feedbacks.append(feedback)
```

```
df = pd.DataFrame({
    'RespondentID': respondents,
    'Feedback': feedbacks
})
```

```
print("Survey Responses with Missing Feedback Filled:")
print(df.to_string(index=False))
```

Status : Correct

Marks : 10/10

### 3. Problem Statement

A software development company wants to classify its employees based on their years of service at the company. They want to categorize employees into three experience levels: Junior (less than 3 years), Mid (3 to 6 years, inclusive), and Senior (more than 6 years).

Experience Level Classification:

Junior: Years at Company  $< 3$

Mid:  $3 \leq$  Years at Company  $< 6$

Senior: Years at Company  $> 6$

You need to create a Python program using the pandas library that reads employee data, processes it into a DataFrame, and adds a new column "Experience Level" to display the appropriate classification for each employee.

#### **Input Format**

First line: an integer  $n$  representing the number of employees.

Next  $n$  lines: each line has a string Name and a floating-point number Years at Company (space-separated).

#### **Output Format**

First line: "Employee Data with Experience Level:"

The employee data table printed with no index column, and with columns: Name, Years at Company, Experience Level.

Refer to the sample output for the formatting specifications.

#### **Sample Test Case**

Input: 5  
Alice 2  
Bob 4  
Charlie 7  
Diana 3  
Evan 6

Output: Employee Data with Experience Level:

Name	Years at Company	Experience Level
Alice	2.0	Junior
Bob	4.0	Mid
Charlie	7.0	Senior
Diana	3.0	Mid
Evan	6.0	Senior

**Answer**

```
import pandas as pd
```

```
n = int(input())  
names = []  
years = []
```

```
for _ in range(n):  
    parts = input().split()  
    names.append(parts[0])  
    years.append(float(parts[1]))
```

```
df = pd.DataFrame({  
    'Name': names,  
    'Years at Company': years  
})
```

```
def classify_experience(years):  
    if years < 3:  
        return 'Junior'  
    elif years <= 5:  
        return 'Mid'  
    else:  
        return 'Senior'
```

```
df['Experience Level'] = df['Years at Company'].apply(classify_experience)
```

```
print("Employee Data with Experience Level:")
```

```
print(df.to_string(index=False))
```

**Status :** Partially correct

**Marks :** 9/10

#### 4. Problem Statement

Arjun is a data scientist working on an image processing task. He needs to normalize the pixel values of a grayscale image matrix to scale between 0 and 1. The input image data is provided as a matrix of integers.

Help him to implement the task using the numpy package.

Formula:

To normalize each pixel value in the image matrix:

$$\text{normalized\_pixel} = (\text{pixel} - \text{min\_pixel}) / (\text{max\_pixel} - \text{min\_pixel})$$

where min\_pixel and max\_pixel are the minimum and maximum pixel values in the image matrix, respectively. If all pixel values are the same, the normalized image matrix should be filled with zeros.

#### **Input Format**

The first line of input consists of an integer value, rows, representing the number of rows in the image matrix.

The second line of input consists of an integer value, cols, representing the number of columns in the image matrix.

The next rows lines each consist of cols integer values separated by a space, representing the pixel values of the image matrix.

#### **Output Format**

The output prints: normalized\_image

Refer to the sample output for the formatting specifications.

#### **Sample Test Case**

Input: 2

3

1 2 3

4 5 6

Output:  $\begin{bmatrix} 0. & 0.2 & 0.4 \\ 0.6 & 0.8 & 1. \end{bmatrix}$

### Answer

```
import numpy as np
```

```
rows = int(input())
```

```
cols = int(input())
```

```
matrix = []
```

```
for _ in range(rows):
```

```
    row = list(map(int, input().split()))
```

```
    matrix.append(row)
```

```
image = np.array(matrix)
```

```
min_pixel = image.min()
```

```
max_pixel = image.max()
```

```
if min_pixel == max_pixel:
```

```
    normalized_image = np.zeros_like(image, dtype=float)
```

```
else:
```

```
    normalized_image = (image - min_pixel) / (max_pixel - min_pixel)
```

```
print(normalized_image)
```

**Status :** Correct

**Marks :** 10/10

## 5. Problem Statement

You're analyzing the daily returns of a set of financial assets over a period of time. Each day is represented as a row in a 2D array, where each column represents the return of a specific asset on that day.

Your task is to identify which days had all positive returns across every

asset using numpy, and output a boolean array indicating these days.

### ***Input Format***

The first line of input consists of two integer values, rows and cols, separated by a space.

Each of the next rows lines consists of cols float values representing the returns of the assets for that day.

### ***Output Format***

The first line of output prints: "Days where all asset returns were positive:"

The second line of output prints: the boolean array positive\_days, indicating True for days where all asset returns were positive and False otherwise.

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 3 4

0.01 0.02 0.03 0.04

0.05 0.06 0.07 0.08

-0.01 0.02 0.03 0.04

Output: Days where all asset returns were positive:

[ True True False]

### ***Answer***

```
import numpy as np
```

```
rows, cols = map(int, input().split())
```

```
data = []
```

```
for _ in range(rows):
```

```
    data.extend(map(float, input().split()))
```

```
array = np.array(data).reshape(rows, cols)
```

```
positive_days = np.all(array > 0, axis=1)
```

```
print("Days where all asset returns were positive:", positive_days)
```



**Status : Correct**

**Marks : 10/10**

# Rajalakshmi Engineering College

Name: UMAR FAROOK

Email: 240701573@rajalakshmi.edu.in

Roll no: 240701573

Phone: 9791730398

Branch: REC

Department: I CSE FF

Batch: 2028

Degree: B.E - CSE

Scan to verify results



## NeoColab\_REC\_CS23221\_Python Programming

### REC\_Python\_Week 7\_CY

Attempt : 1

Total Mark : 50

Marks Obtained : 50

### Section 1 : Coding

#### 1. Problem Statement

Arjun is monitoring hourly temperature data recorded continuously for multiple days. He needs to calculate the average temperature for each day based on 24 hourly readings.

Help him to implement the task using the numpy package.

Formula:

Reshape the temperature readings into rows where each row has 24 readings (one day).

Average temperature per day = mean of 24 hourly readings in each row.

**Input Format**

The first line of input consists of an integer value, n, representing the total number of temperature readings.

The second line of input consists of n floating-point values separated by spaces, representing hourly temperature readings.

### **Output Format**

The output prints: avg\_per\_day

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 30

30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0  
30.0 30.0 30.0 30.0 30.0 30.0 30.0 30.0

Output: [30.]

### **Answer**

```
import numpy as np
```

```
n = int(input())
```

```
readings = list(map(float, input().split()))
```

```
data = np.array(readings).reshape(-1, 24)
```

```
avg_per_day = data.mean(axis=1)
```

```
print(avg_per_day)
```

**Status :** Correct

**Marks :** 10/10

## **2. Problem Statement**

You are working as a data analyst for a small retail store that wants to track the stock levels of its products. Each product has a unique Name (such as "Toothpaste", "Shampoo", "Soap") and an associated Quantity in stock. Management wants to identify which products have zero stock so they can be restocked.

Write a Python program using the pandas library to help with this task. The program should:

Read the number of products, n. Read n lines, each containing the Name of the product and its Quantity, separated by a space. Convert this data into a pandas DataFrame. Identify and display the Name and Quantity of products with zero stock. If no products have zero stock, display: No products with zero stock.

### ***Input Format***

The first line contains an integer n, the number of products.

The next n lines each contain:

<Product\_ID> <Quantity>

where <Product\_ID> is a single word (e.g., "Shampoo") and <Quantity> is a non-negative integer (e.g., 5).

### ***Output Format***

The first line of output prints:

Products with Zero Stock:

If there are any products with zero stock, the following lines print the pandas DataFrame showing those products with two columns: Product\_ID and Quantity.

The column headers Product\_ID and Quantity are printed in the second line.

Each subsequent line shows the product's name and quantity, aligned under the respective headers, with no index column.

The output formatting (spacing and alignment) follows the default pandas `to_string(index=False)` style.

If no products have zero stock, print:

No products with zero stock.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 3

P101 10

P102 0

P103 5

Output: Products with Zero Stock:

Product_ID	Quantity
------------	----------

P102	0
------	---

### **Answer**

```
import pandas as pd
```

```
n = int(input())
```

```
products = []
```

```
for _ in range(n):
```

```
    name, quantity = input().split()
```

```
    products.append([name, int(quantity)])
```

```
df = pd.DataFrame(products, columns=['Product_ID', 'Quantity'])
```

```
zero_stock = df[df['Quantity'] == 0]
```

```
print("Products with Zero Stock:")
```

```
if zero_stock.empty:
```

```
    print("No products with zero stock.")
```

```
else:
```

```
    print(zero_stock.to_string(index=False))
```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

Arjun is developing a system to monitor environmental sensors installed in different rooms of a smart building. Each sensor records multiple temperature readings throughout the day. To compare sensor data fairly despite differing scales, Arjun needs to normalize each sensor's readings so that they have a mean of zero and standard deviation of one.

Help him implement this normalization using numpy.

Normalization Formula:

#### ***Input Format***

The first line of input consists of two integers: sensors (number of sensors) and samples (number of readings per sensor).

The next sensors lines each contain samples space-separated floats representing the sensor readings.

#### ***Output Format***

The first line of output prints: "Normalized Sensor Data."

The next lines print the normalized readings as a numpy array, where each row corresponds to a sensor's normalized values.

Refer to the sample output for the formatting specifications.

#### ***Sample Test Case***

Input: 3 3  
1.0 2.0 3.0  
4.0 5.0 6.0

7.0 8.0 9.0

Output: Normalized Sensor Data:

```
[[ -1.22474487  0.      1.22474487]
 [ -1.22474487  0.      1.22474487]
 [ -1.22474487  0.      1.22474487]]
```

**Answer**

```
import numpy as np
```

```
sensors, samples = map(int, input().split())
data = []
```

```
for _ in range(sensors):
    readings = list(map(float, input().split()))
    data.append(readings)
```

```
arr = np.array(data)
means = arr.mean(axis=1, keepdims=True)
stds = arr.std(axis=1, keepdims=True)
normalized = (arr - means) / stds
```

```
print("Normalized Sensor Data:")
print(normalized)
```

**Status : Correct**

**Marks : 10/10**

#### 4. Problem Statement

Rekha is a meteorologist analyzing rainfall data collected over 5 years, with monthly rainfall recorded for each year. She wants to find the total rainfall each year and also identify the month with the maximum rainfall for every year.

Help her to implement the task using the numpy package.

Formula:

Yearly total rainfall = sum of all 12 months' rainfall for each year

Month with max rainfall = index of the maximum rainfall value within the

12 months for each year (0-based index)

### ***Input Format***

The input consists of 5 lines.

Each line contains 12 floating-point values separated by spaces, representing the rainfall data (in mm) for each month of that year.

### ***Output Format***

The first line of output prints: yearly\_totals

The second line of output prints: max\_rainfall\_months

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0  
2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0  
3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0  
4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0  
5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0

Output: [ 78. 90. 102. 114. 126.]  
[11 11 11 11 11]

### ***Answer***

```
import numpy as np

data = []
for _ in range(5):
    row = list(map(float, input().split()))
    data.append(row)

arr = np.array(data)

yearly_totals = arr.sum(axis=1)
max_rainfall_months = arr.argmax(axis=1)

print(yearly_totals)
```



```
print(max_rainfall_months)
```

**Status :** Correct

**Marks :** 10/10

## 5. Problem Statement

Rekha works as an e-commerce data analyst. She receives transaction data containing purchase dates and needs to extract the month and day from these dates using the pandas package.

Help her implement this task by performing the following steps:

Convert the Purchase Date column to datetime format, treating invalid date entries as NaT (missing).

Create two new columns:

Purchase Month, containing the month (as an integer) extracted from the Purchase Date.

Purchase Day, containing the day (as an integer) extracted from the Purchase Date. Keep the rest of the data as is.

### **Input Format**

The first line of input contains an integer  $n$ , representing the number of records.

The second line contains the CSV header — comma-separated column names.

The next  $n$  lines each contain a transaction record in comma-separated format.

### **Output Format**

The first line of output is the text:

Transformed E-commerce Transaction Data:

The next lines print the pandas DataFrame with:

The original columns (including Purchase Date, which is now in datetime format or NaT if invalid).

Two additional columns: Purchase Month and Purchase Day.

The output uses the default pandas DataFrame string representation as produced by `print(transformed_df)`.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 3

Customer,Purchase Date

Alice,2023-05-15

Bob,2023-06-20

Charlie,2023-07-01

Output: Transformed E-commerce Transaction Data:

	Customer	Purchase Date	Purchase Month	Purchase Day
0	Alice	2023-05-15	5	15
1	Bob	2023-06-20	6	20
2	Charlie	2023-07-01	7	1

### **Answer**

```
import pandas as pd
import sys
n = int(input())
header = input().split(',')

data = []
for _ in range(n):
    row = input().split(',')
    data.append(row)
df = pd.DataFrame(data, columns=header)
df['Purchase Date'] = pd.to_datetime(df['Purchase Date'], errors='coerce')
df['Purchase Month'] = df['Purchase Date'].dt.month
df['Purchase Day'] = df['Purchase Date'].dt.day
print("Transformed E-commerce Transaction Data:")
print(df)
```

**Status :** Correct

**Marks :** 10/10