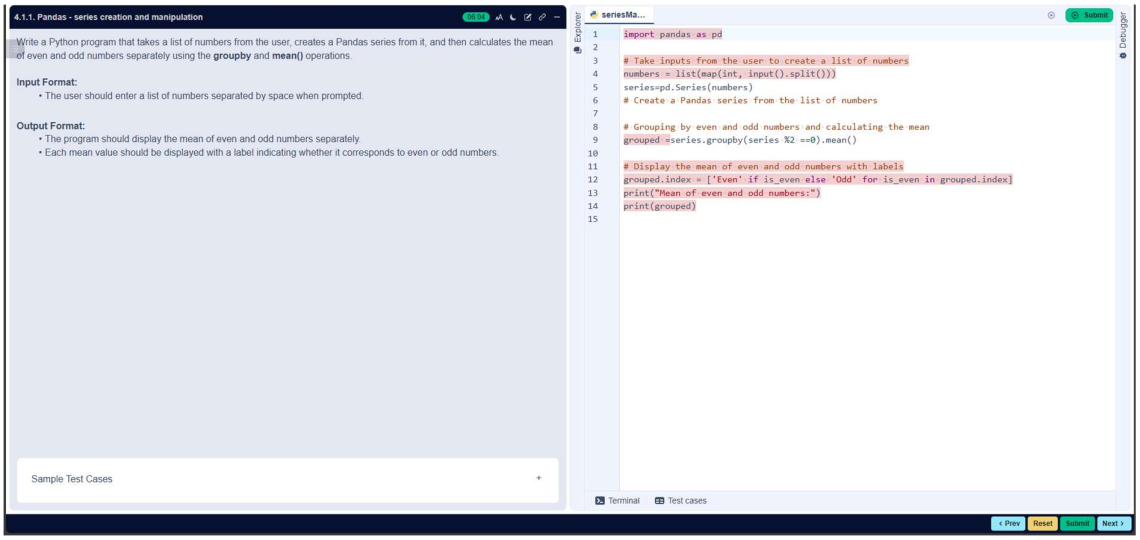
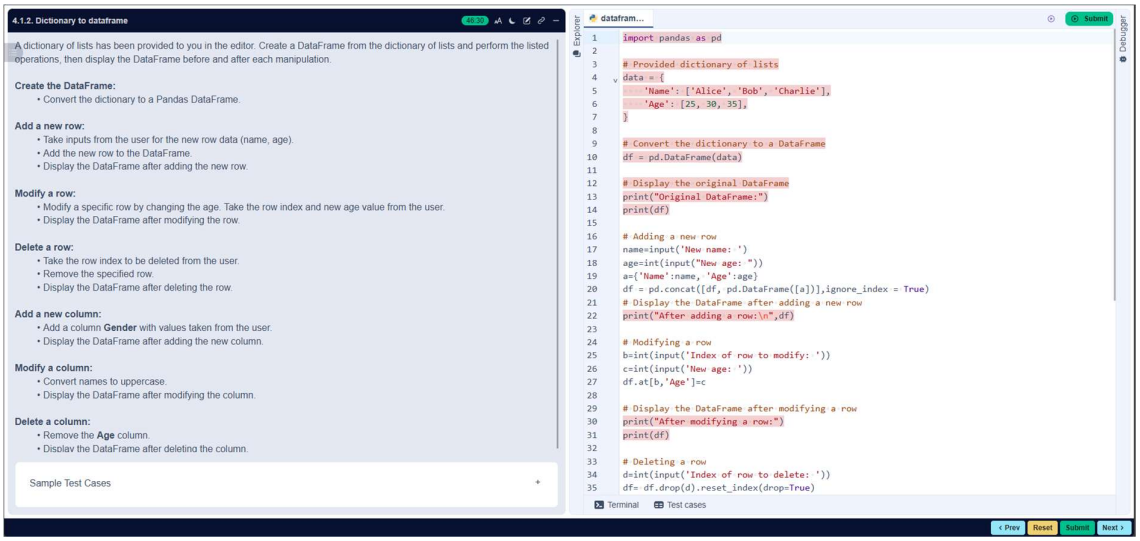
Name : Yash Kundlik Jambhale.

Div : CS1-09,C11.

Sub : EDS LAB(CodeTantra)





4.1.3. Student Information

15:16

Write a program to read a text file containing student information (name, age, and grade) using Pandas. Perform the following tasks:

- Display the first five rows of the data frame.
- Calculate the average age of the students(limit the average age up to 2 decimal places).
- Filter out the students who have a grade above a certain threshold(consider the threshold grade is **'B'**).

Note:
Refer to the displayed test cases for better understanding.

Sample Test Cases +

studentin... studentdat... Submit

```python
import pandas as pd

# Read the text file into a DataFrame
file = input()
data = pd.read_csv(file, sep="\s+", header=None, names=["Name", "Age", "Grade"])
print('First five rows:')
print(data.head())
avg=round(data['Age'].mean(),2)
print("Average age:",avg)
# write your code here..

print('Students with a grade up to B')
filtred=data[data['Grade'] <= 'B']
print(filtred)
```

Terminal Test cases

Explorer Debugger

‹ Prev Reset Submit Next ›

4.2.1. Month with the Highest Total Sales

01:05

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- The CSV file contains the columns: `Date, Product, Quantity, Price,` and `City`.
- Group the data by Month and calculate the total sales for each month.
- Find the month with the highest total sales and display it.
- Also, display the total sales for the best month.

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
2025-01-05,Product C,10,30,Los Angeles
```

Note:
The data cannot be displayed in the file. You can refer to the sample data provided for insights.

Sample Test Cases +

monthFor... sales_dat... Submit

```python
import pandas as pd

# Prompt the user for the file name
file_name = input()

# Load the data
df = pd.read_csv(file_name)

df['Date'] = pd.to_datetime(df['Date'])

# Extract month in 'YYYY-MM' format
df['Month'] = df['Date'].dt.to_period('M')

# Calculate total sales (Quantity * Price)
df['Total Sales'] = df['Quantity'] * df['Price']

# Group by month and sum total sales
monthly_sales = df.groupby('Month')['Total Sales'].sum()

# Find the month with the highest total sales
best_month = monthly_sales.idxmax()  # Get the month with max sales
highest_sales = monthly_sales.max()  # Get the max
# Find the month with the highest total sales

print(f"Best month: {best_month}")
print(f"Total sales: ${highest_sales:.2f}")

```

Terminal Test cases

Explorer Debugger

‹ Prev Reset Submit Next ›

4.2.2. Best Selling Product

02:12

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- The CSV file contains the columns: `Date, Product, Quantity, Price,` and `City`.
- Find the product that sold the most in terms of quantity sold.
- Display the product that sold the most and the total quantity sold for that product.

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
2025-01-05,Product C,10,30,Los Angeles
```

Note:
The data cannot be displayed in the file. You can refer to the sample data provided for insights.

Sample Test Cases +

monthFor... sales_dat... Submit

```python
import pandas as pd

# Prompt the user for the file name
file_name = input()

# Load the data
df = pd.read_csv(file_name)


# Find the product with the highest total quantity sold
product_sales = df.groupby("Product")["Quantity"].sum()
best_product = product_sales.idxmax()
highest_quantity = product_sales.max()

# Display the result
print(f"Best selling product: {best_product}")
print(f"Total quantity sold: {highest_quantity}")

```

Terminal Test cases

Explorer Debugger

‹ Prev Reset Submit Next ›

4.2.4. Most Frequently Sold Product Pairs

Write a Python program that takes the file name of a CSV file as input, reads the data, and performs the following operations:

- The CSV file contains the following columns: Date, Product, Quantity, Price, and City.
- For each date, find all pairs of products that were sold together (i.e., two products sold on the same date).
- Output the product pair/s that was sold most frequently.

Sample Data:

```
Date,Product,Quantity,Price,City
2025-01-01,Product A,5,20,New York
2025-01-01,Product B,3,15,Los Angeles
2025-01-02,Product A,7,20,New York
2025-01-02,Product C,4,30,Chicago
2025-01-03,Product B,2,15,Chicago
2025-01-03,Product A,8,20,Los Angeles
2025-01-04,Product C,6,30,New York
2025-01-04,Product B,5,15,Los Angeles
2025-01-05,Product A,3,20,Chicago
2025-01-05,Product C,10,30,Los Angeles
```

Explanation:
Transactions:

- **2025-01-01:** Product A, Product B
- **2025-01-02:** Product A, Product C
- **2025-01-03:** Product B, Product A
- **2025-01-04:** Product C, Product B
- **2025-01-05:** Product A, Product C

Now, let's count how often the pairs of products appear together:

- **Product A and Product B**: Appear in transactions on 2025-01-01 and 2025-01-03.
- **Product A and Product C**: Appear in transactions on 2025-01-02 and 2025-01-05.
- **Product B and Product C**: Appears in transactions on 2025-01-04.

Most Frequent Product Combinations:

- **Product A and Product B** (2 times)
- **Product A and Product C** (2 times)

Sample Test Cases +

frequenti... sales_dat... Submit

```python
import pandas as pd
from itertools import combinations
from collections import Counter

# Prompt user to input the file name
file_name = input()

# Read data from the specified CSV file
df = pd.read_csv(file_name)

# write the code


date_product_dict = df.groupby("Date")["Product"].apply(list).to_dict()

# Generate product pairs for each date
pair_counts = Counter()
for products in date_product_dict.values():
    for pair in combinations(sorted(products), 2):  # Sorting ensures consistent pair order
        pair_counts[pair] += 1

# Find the most frequent product pair(s)
max_count = max(pair_counts.values())
most_frequent_pairs = {pair: count for pair, count in pair_counts.items() if count == max_count}

# Output the most frequent product pairs
for (product1, product2), count in most_frequent_pairs.items():
    print(f"{product1} and {product2}: {count} times")
```

Terminal Test cases

‹ Prev Reset Submit Next ›

5.1.1. Stacked Plot

Create a stacked area plot to visualize the temperature variations for three different cities (City A, City B, and City C) across the months of the year. The temperature data is provided for each city in the editor.

Your task is to:

- Create a stacked area plot using the data.
- Label the x-axis as "Month", the y-axis as "Temperature", and provide the title "Temperature Variation" for the plot.
- Display the plot showing the temperature variation for each city throughout the months of the year.

Sample Test Cases +

stackedpl... Submit

```python
import matplotlib.pyplot as plt
import pandas as pd

# Data for Months and Temperature for three cities
data = {
    'Month': ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'],
    'City_A_Temperature': [5, 7, 10, 13, 17, 20, 22, 21, 18, 12, 8, 6],
    'City_B_Temperature': [2, 3, 5, 6, 10, 14, 16, 17, 12, 9, 5, 3],
    'City_C_Temperature': [3, 4, 6, 8, 9, 12, 15, 14, 10, 7, 4, 2]
}
df=pd.DataFrame(data)
plt.stackplot(df['Month'],df['City_A_Temperature'],df['City_B_Temperature'],df['City_C_Temperature'])

plt.title('Temperature Variation')
plt.ylabel('Temperature')
plt.xlabel('Month')
plt.show()
```

Terminal Test cases

‹ Prev Reset Submit Next ›

1.2.1. Pass or Fail

Write a Python program that accepts the number of courses and the marks of a student in those courses.
The grade is determined based on the aggregate percentage:

- If the aggregate percentage is greater than 75, the grade is Distinction.
- If the aggregate percentage is greater than or equal to 60 but less than 75, the grade is First Division.
- If the aggregate percentage is greater than or equal to 50 but less than 60, the grade is Second Division.
- If the aggregate percentage is greater than or equal to 40 but less than 50, the grade is Third Division.

Input Format:
The first input will be an integer n, the number of courses.
The second input will be n integers representing the marks of the student in each of the n courses, separated by a space.

Output Format:
If the student passes all courses:

- Print the aggregate percentage (rounded to two decimal places).
- Print the grade based on the aggregate percentage.

If the student fails any course (marks < 40 in any course), print:

- "Fail".

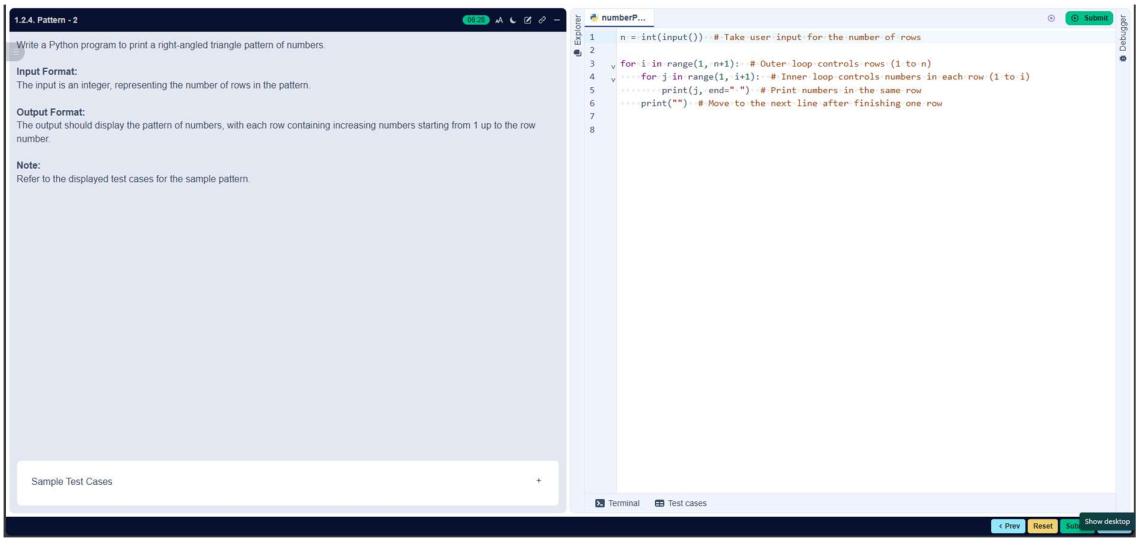
Sample Test Cases +

passorFa... Submit

```python
n=int(input())
marks=input().split()
intlist=[int(m) for m in marks]
t=0
count=0
for m in intlist:
    if m>=40:
        t=t+m
    else:
        count=count+1
tm=n*100
if count==0:
    p=(t/tm)*100
    if p>=75:
        print("Aggregate Percentage:",'%.2f'%p)
        print("Grade: Distinction")
    elif p>=60:
        print("Aggregate Percentage:",'%.2f'%p)
        print("Grade: First Division")
    elif p>=50:
        print("Aggregate Percentage:",'%.2f'%p)
        print("Grade: Second Division")
    elif p>=40:
        print("Aggregate Percentage:",'%.2f'%p)
        print("Grade: Third Division")
else:
    print("Fail")
```

Terminal Test cases

‹ Prev Reset Submit Next ›



1.2.4. Pattern - 2

Write a Python program to print a right-angled triangle pattern of numbers.

Input Format:
The input is an integer, representing the number of rows in the pattern.

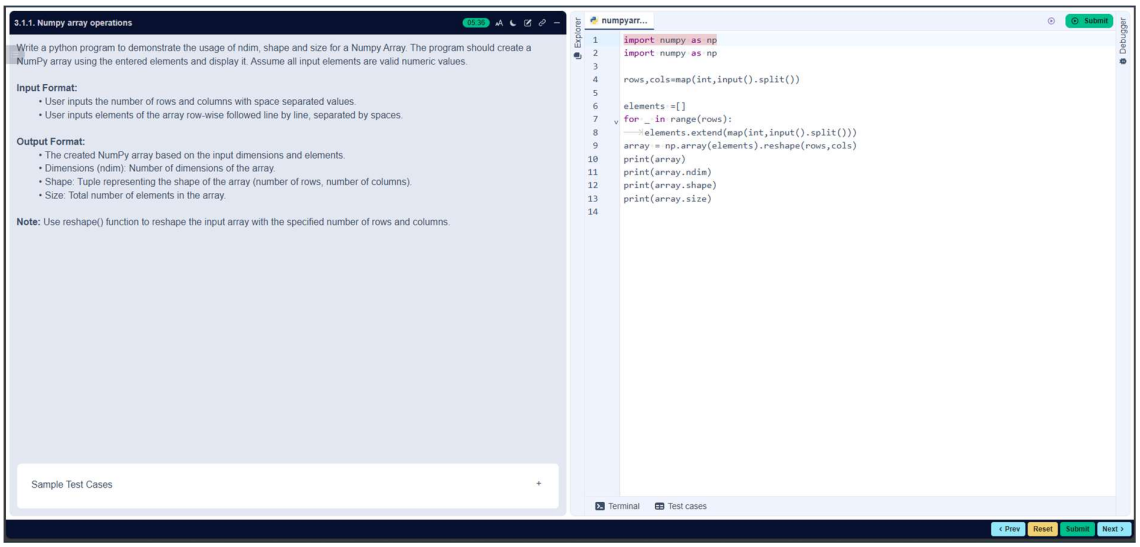
Output Format:
The output should display the pattern of numbers, with each row containing increasing numbers starting from 1 up to the row number.

Note:
Refer to the displayed test cases for the sample pattern.

Sample Test Cases +

```
n = int(input())  # Take user input for the number of rows

for i in range(1, n+1):  # Outer loop controls rows (1 to n)
    for j in range(1, i+1):  # Inner loop controls numbers in each row (1 to i)
        print(j, end=" ")  # Print numbers in the same row
    print("")  # Move to the next line after finishing one row
```



3.1.1. Numpy array operations

Write a python program to demonstrate the usage of ndim, shape and size for a Numpy Array. The program should create a NumPy array using the entered elements and display it. Assume all input elements are valid numeric values.

Input Format:
- User inputs the number of rows and columns with space separated values.
- User inputs elements of the array row-wise followed line by line, separated by spaces.

Output Format:
- The created NumPy array based on the input dimensions and elements.
- Dimensions (ndim): Number of dimensions of the array.
- Shape: Tuple representing the shape of the array (number of rows, number of columns).
- Size: Total number of elements in the array.

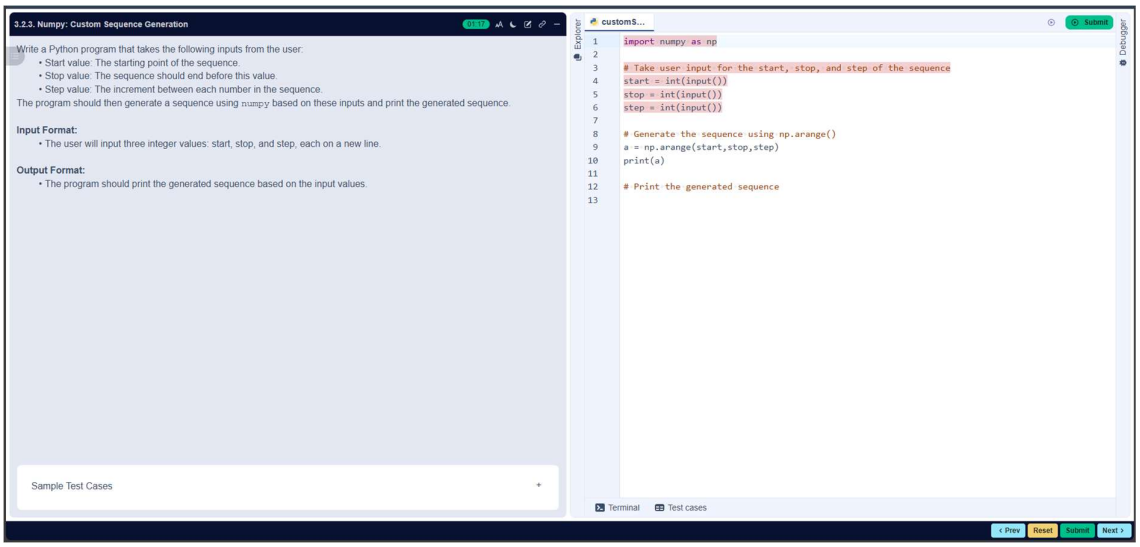
Note: Use reshape() function to reshape the input array with the specified number of rows and columns.

Sample Test Cases +

```
import numpy as np
import numpy as np

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

elements =[]
for _ in range(rows):
    elements.extend(map(int,input().split()))
array = np.array(elements).reshape(rows,cols)
print(array)
print(array.ndim)
print(array.shape)
print(array.size)
```



3.2.3. Numpy: Custom Sequence Generation

Write a Python program that takes the following inputs from the user:
- Start value: The starting point of the sequence.
- Stop value: The sequence should end before this value.
- Step value: The increment between each number in the sequence.

The program should then generate a sequence using `numpy` based on these inputs and print the generated sequence.

Input Format:
- The user will input three integer values: start, stop, and step, each on a new line.

Output Format:
- The program should print the generated sequence based on the input values.

Sample Test Cases +

```
import numpy as np

# Take user input for the start, stop, and step of the sequence
start = int(input())
stop = int(input())
step = int(input())

# Generate the sequence using np.arange()
a = np.arange(start,stop,step)
print(a)

# Print the generated sequence
```