**Loop Examples in Python**

**Run the following codes and discuss with your friends what they produces.**

**#1. Loop through a List**  
fruits = ['apple', 'banana', 'cherry']  
for fruit in fruits:  
 print(fruit)  
**#2. Loop with range()**  
for i in range(5):  
 print(i)  
**#3. While loop**  
i = 0  
while i < 5:  
 print(i)  
 i += 1  
**#4. Loop through a String**  
for char in 'hello':  
 print(char)  
**#5. Break in a For Loop**  
for i in range(10):  
 if i == 3:  
 break  
print(i)  
**#6. Continue in a For Loop**  
for i in range(5):  
 if i == 2:  
 continue  
 print(i)  
**#7. While loop with break**  
i = 0  
while True:  
 if i == 4:  
 break  
 print(i)  
 i += 1

**#8. While loop with continue**  
i = 0  
while i < 5:  
 i += 1  
 if i == 3:  
 continue  
 print(i)  
**#9. For-else Loop**  
for i in range(3):  
 print(i)  
else:  
 print('Loop completed.')  
**#10. Nested For Loops**  
for i in range(1, 4):  
 for j in range(1, 4):  
 print(f'{i} \* {j} = {i \* j}')  
**#11. Using enumerate()**  
colors = ['red', 'green', 'blue']  
for index, color in enumerate(colors):  
 print(index, color)  
**#12. Loop through Dictionary**  
person = {'name': 'Alice', 'age': 25}  
for key, value in person.items():  
 print(f'{key}: {value}')  
**#13. Using zip()**  
names = ['Anna', 'Ben']  
scores = [90, 85]  
for name, score in zip(names, scores):  
 print(f'{name} scored {score}')  
**#14. Loop in Reverse**  
for i in reversed(range(5)):  
 print(i)  
  
**#15. Loop with sorted()**  
nums = [3, 1, 4, 2]  
for n in sorted(nums):  
 print(n)  
**#16. List Comprehension (Squares)**  
squares = [x\*\*2 for x in range(5)]  
print(squares)

**#17. Conditional List Comprehension**evens = [x for x in range(10) if x % 2 == 0]  
print(evens)  
**#18. Set Comprehension**  
unique\_letters = {letter for letter in 'banana'}  
print(unique\_letters)  
**#19. Dictionary Comprehension**  
cubes = {x: x\*\*3 for x in range(4)}  
print(cubes)  
**#20. Count Vowels in Text**  
text = 'Hello World'  
vowels = 'aeiouAEIOU'  
for ch in text:  
 if ch in vowels:  
 print(f'Vowel found: {ch}')

**Questions**

**#Question: Create a program that reminds the user to drink water 5 times a day using a loop.**  
**# Program 1: Drink Water Reminder (5 times)**  
for reminder in range(1, 6):  
 print(f"Reminder {reminder}: Drink Water")  
  
**#Question: Remind user to drink water every 2 hours from 6 AM to 6 PM.  
# Program 2: Drink Water Reminder between 6 AM and 6 PM**  
hour = 6  
for reminder in range(1, 7):  
 print(f"Reminder {reminder} at {hour}:00 - Drink Water")  
 hour += 2  
  
**#Question: Display a message one character at a time (like typing).  
# Program 3: Typing Animation**  
message = "Welcome to the Game!"  
for ch in message:  
 print(ch, end='', flush=True)

**#Question: Stop searching when “milk” is found in the inventory.  
# Program 4: Item Search in Warehouse**  
  
inventory = ["bread", "eggs", "milk", "butter"]  
  
for item in inventory:  
 print(f"Checking: {item}")  
 if item == "milk":  
 print("✅ Item found:", item)  
 break  
  
**#Question: Display all available items, skip the one marked ‘unavailable’.  
# Program 5: Skip Unavailable Items**items = ["laptop", "mouse", "unavailable", "keyboard", "monitor"]  
  
print("Available items:")  
for item in items:  
 if item == "unavailable":  
 continue  
 print("-", item)  
  
**#Question: Simulate traffic light waiting until it turns green.  
# Program 6: Traffic Light Countdown**  
for second in range(1, 6):  
 print(f"Red Light... {second} second(s)")  
print("✅ Green Light!")  
  
**#Question: Simulate elevator stopping from 1–20, skipping floor 13.  
# Program 7: Elevator skipping 13th floor**  
for floor in range(1, 21):  
 if floor == 13:  
 continue  
 print(f"Elevator stopping at floor {floor}")  
  
**#Question: Countdown from 4 to 0 for a rocket launch.  
# Program 8: Rocket Launch Countdown**  
for count in range(4, -1, -1):  
 print(count)  
print("Lift Off!")

**Defining Functions**

def fun(name):  
 print(f"Hi, {name}")  
  
  
fun("Ali")  
  
  
def equation(x, y):  
 return (x \* 2 + y \* 2 + 2 \* x \* y)  
  
  
equation(3, 4)  
equation(-3, 2)  
  
  
def sq(x):  
 return x \* x  
  
  
sq(5)  
  
  
def cal(x, y):  
 return x + y, x \* y  
  
  
cal(10, 15)  
  
  
def rect\_a\_p(length, width):  
 area = length \* width  
 perimeter = 2 \* (length + width)  
 return area, perimeter  
  
  
rect\_a\_p(2, 6)  
  
length = 5  
width = 2  
area, perimeter = rect\_a\_p(length, width)  
print(f"area : {area}")  
print(f"perimeter : {perimeter}")  
  
import math  
  
  
def area\_circle(radius):  
 return math.pi \* radius \* 2  
  
  
area\_circle(3)  
  
  
def even(n):  
 return n % 2 == 0  
  
  
print(even(4))  
print(even(7))  
  
  
def m(a, b):  
 return a if a > b else b  
  
  
print(m(7, 12))  
  
  
def fact(n):  
 if n == 0:  
 return 1  
 return n \* fact(n - 1)  
  
  
print(fact(5))  
  
  
def count\_vowels(text):  
 vowels = "aeiouAEIOU"  
 return sum(1 for char in text if char in vowels)  
  
  
print(count\_vowels("Statistics and Data Science"))  
  
print(count\_vowels("Jahangirnagar University"))  
  
  
def rev\_string(st):  
 return st[::-1]  
  
  
print(rev\_string("python"))  
  
  
def count\_words(sentence):  
 return len(sentence.split())  
  
  
print(count\_words("More para in the Dept of SDS"))  
  
  
def count\_charecter(text, ch):  
 return text.count(ch)  
  
  
print(count\_charecter("mango", "a"))  
print(count\_charecter("Jahangirnagar", "a"))  
  
  
def sum\_list(lst):  
 return sum(lst)  
  
  
print(sum\_list([1, 2, 3, 4, 5]))  
  
  
def largest(lst):  
 return max(lst)  
  
  
print(largest([4, 10, 20, 50, 34]))  
  
  
def ave(nums):  
 return sum(nums) / len(nums)  
  
  
print(ave([10, 20, 30]))  
  
  
def check\_number(n):  
 if n > 0:  
 return "Positive"  
 elif n < 0:  
 return "Negaive"  
 else:  
 return "Zero"  
  
  
print(check\_number(-5))  
print(check\_number(5))  
print(check\_number(0))

**Mean & Variance**

import numpy as np  
data=np.array([5,15,10,20,25,30,25,35])  
  
**#Mean**  
mean=np.mean(data)  
print('Mean:',mean)  
  
**#Median**  
median=np.median(data)  
print('Median:',median)  
  
**#Mode**  
from scipy import stats  
mode=stats.mode(data)  
print("Mode:",mode[0])  
**#Show dictionary**  
dir(np)  
import scipy as sc  
dir(sc)  
**#SD**  
std\_dev=np.std(data)  
print("Standard Deviation:",std\_dev)  
  
**#Variance**  
variance=np.var(data)  
print('Variance',variance)  
  
**#Skewnwess ,Kurtosis**  
from scipy import stats  
Skewness=stats.skew(data)  
Kurtosis=stats.kurtosis(data)  
print('Skewness:',Skewness)  
print('Kurtosis:',Kurtosis)  
  
X=np.array([1,2,3,4,5])  
Y=np.array([2,4,5,4,5])  
correlation\_matrix=np.corrcoef(X,Y)  
correlation= correlation\_matrix[0,1]  
print("Pearson correlation coefficient:",correlation)  
mean\_X= np.mean(X)  
mean\_Y= np.mean(Y)

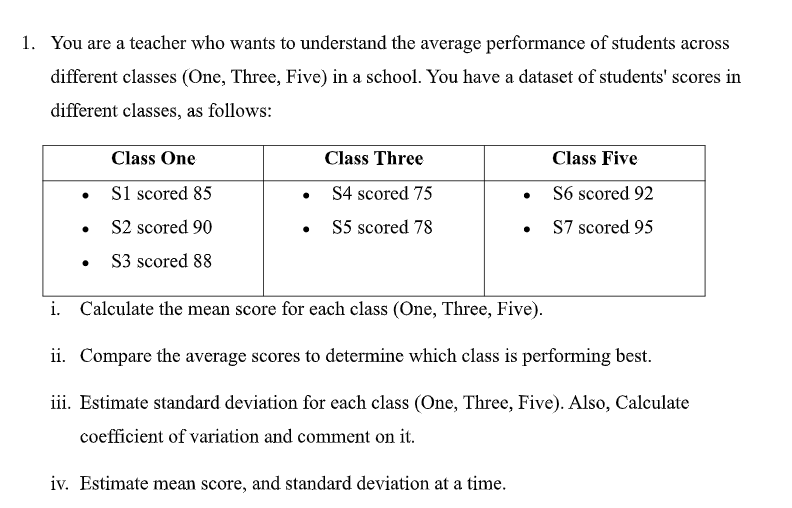
**Diagrams**

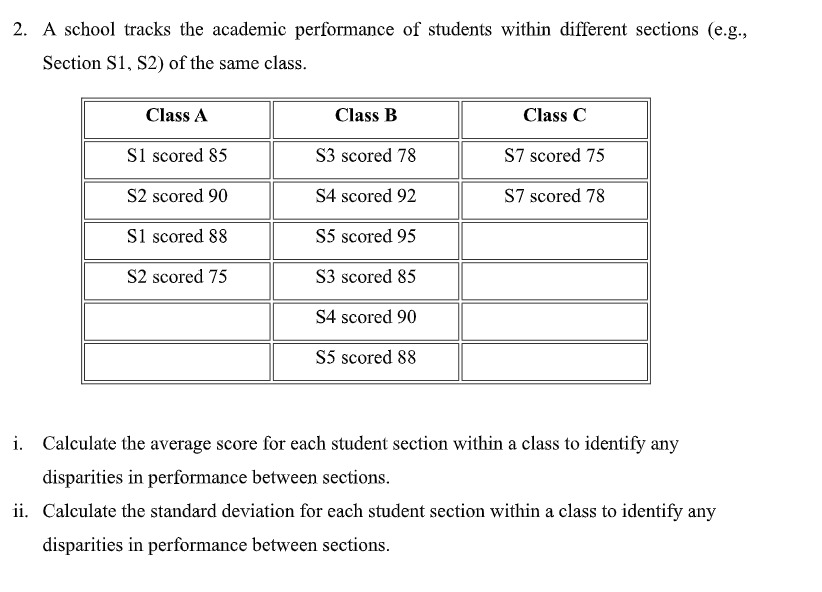
import matplotlib.pyplot as plt  
  
**# Sample data**  
x = [1, 2, 3, 4, 5]  
y = [5, 7, 4, 6, 8]  
  
**# Create scatter plot**  
plt.scatter(x, y, color='blue', marker='v', label='Data Points')  
  
**# Connect points with a line**  
plt.plot(x, y, color='red', linestyle='-', linewidth=1, label='Connecting Line')  
  
**# Add value labels for each point**  
for i in range(len(x)):  
 plt.text(x[i], y[i], f'({x[i]},{y[i]})', fontsize=9, ha='left', va='top')  
  
**# Add labels, title, and legend**  
plt.xlabel('X Axis')  
plt.ylabel('Y Axis')  
plt.title('Scatter Plot with Connecting Line')  
plt.legend()  
  
**# Show plot**  
plt.show()  
  
**#####two scatter line in a plot**  
import matplotlib.pyplot as plt  
  
**# Data**  
x = [1, 2, 3, 4, 5]  
y = [5, 7, 4, 6, 8]  
z = [2, 4, 6, 7, 8]  
  
**# Plot x vs y**  
plt.scatter(x, y, color='blue', marker='o', label='x vs y')  
plt.plot(x, y, color='blue', linestyle='-', linewidth=1)  
  
**# Plot x vs z**  
plt.scatter(x, z, color='red', marker='^', label='x vs z')  
plt.plot(x, z, color='red', linestyle='--', linewidth=1)  
  
**# Labels and title**  
plt.xlabel('x')  
plt.ylabel('Values')  
plt.title('Scatter Plot with Two Lines')  
plt.legend()  
  
plt.show()  
  
**###Histogram**  
  
import matplotlib.pyplot as plt  
import numpy as np  
import math  
  
**# Sample data**  
data = [12, 15, 13, 17, 19, 22, 22, 23, 25, 26, 26, 29, 30, 33, 35]  
  
**# Calculate bins using Sturges' formula**  
n = len(data)  
bins = math.ceil(np.log2(n) + 1)  
  
**# Plot histogram and get counts and bin edges**counts, bin\_edges, patches = plt.hist(data, bins=bins, color='skyblue', edgecolor='black')  
  
**# Calculate bin midpoints for labeling**  
bin\_mids = (bin\_edges[:-1] + bin\_edges[1:]) / 2  
  
  
**# Add axis labels and title**  
plt.xlabel('Value')  
plt.ylabel('Frequency')  
plt.title(f'Histogram with bin midpoints labeled (bins={bins})')  
  
**# Set new x-axis labels**  
plt.xticks(bin\_mids, [f'{mid:.1f}' for mid in bin\_mids])  
plt.show()  
  
  
  
  
**#### Bar Diagram**  
import matplotlib.pyplot as plt  
from collections import Counter  
  
**# Your categorical data**data = ['A', 'A', 'B', 'A', 'C', 'A', 'B', 'C', 'D', 'E', 'D', 'F', 'E', 'F', 'C', 'D', 'A', 'F', 'A']  
  
**# Count frequency of each category**  
freq = Counter(data)  
  
**# Separate the categories and their counts**categories = list(freq.keys())  
values = list(freq.values())  
  
**# Create bar plot**  
plt.bar(categories, values, color='skyblue', edgecolor='red')

**# Add labels and title**  
plt.xlabel('Category')  
plt.ylabel('Frequency')  
plt.title('Bar Diagram of Categorical Data')  
  
**# Show plot**  
plt.show()  
  
**#### Pie chart**  
import matplotlib.pyplot as plt  
from collections import Counter  
  
**# Your categorical data**  
data = ['A', 'A', 'B', 'A', 'C', 'A', 'B', 'C', 'D', 'E', 'D', 'F', 'E', 'F', 'C', 'D', 'A', 'F', 'A']  
  
**# Count frequency of each category**  
freq = Counter(data)  
  
**# Separate the categories and their counts**  
labels = list(freq.keys())  
sizes = list(freq.values())  
  
**# Create pie chart**  
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=plt.cm.Pastel1.colors)  
  
**# Equal aspect ratio ensures pie is a circle**plt.axis('equal')  
plt.title('Pie Chart of Categorical Data')  
  
**# Show plot**  
plt.show()  
  
  
import matplotlib.pyplot as plt  
  
**# Data**  
x = [1, 2, 3, 4, 5]  
y = [5, 7, 4, 6, 8]  
z = [2, 4, 6, 7, 8]  
**# Plot x vs y**  
plt.scatter(x, y, color='blue', marker='o', label='x vs y')  
plt.plot(x, y, color='blue', linestyle='-', linewidth=1)  
**# Plot x vs z**  
plt.scatter(x, z, color='red', marker='^', label='x vs z')  
plt.plot(x, z, color='red', linestyle='--', linewidth=1)  
  
**# Labels and title**  
plt.xlabel('x')  
plt.ylabel('Values')  
plt.title('Scatter Plot with Two Lines')  
plt.legend()  
  
plt.show()  
  
  
 **Mean & SD of Table**  
**#####Mean by Class**  
import pandas as pd  
  
# Sample data  
data = {  
 'Class': ['A', 'A', 'A', 'B', 'B', 'C', 'C'],  
 'Student': ['S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7'],  
 'Score': [85, 90, 88, 75, 78, 92, 95]  
}  
  
df = pd.DataFrame(data)  
print("Original Data:\n", df)  
  
df['Mean by Group'] = df.groupby('Class')['Score'].transform(lambda x: x.mean())  
print("\nMean within each class:\n", df)  
  
  
**######Standard deviation by Class**  
import pandas as pd  
  
# Sample data  
data = {  
 'Class': ['A', 'A', 'A', 'B', 'B', 'C', 'C'],  
 'Student': ['S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7'],  
 'Score': [85, 90, 88, 75, 78, 92, 95]  
}  
  
df = pd.DataFrame(data)  
print("Original Data:\n", df)  
  
df['Standard deviation by Group'] = df.groupby('Class')['Score'].transform(lambda x: x.std())  
print("\nSD within each class:\n", df)  
  
  
  
**######Mean and Standard deviation by Class**  
import pandas as pd  
  
# Sample data  
data = {  
 'Class': ['A', 'A', 'A', 'B', 'B', 'C', 'C'],  
 'Student': ['S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7'],  
 'Score': [85, 90, 88, 75, 78, 92, 95]  
}  
  
df = pd.DataFrame(data)  
print("Original Data:\n", df)  
  
df['Standard deviation by Group'] = df.groupby('Class')['Score'].transform(lambda x: x.std())  
df['Mean by Class'] = df.groupby('Class')['Score'].transform(lambda x: x.mean())  
print("\nSD and Mean within each class:\n", df)  
  
  
  
**######Mean by Class and Student Section**  
import pandas as pd  
  
  
data = {  
 'Class': ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'C', 'C'],  
 'section': ['S1', 'S2','S1', 'S2', 'S3', 'S4', 'S5', 'S3', 'S4','S5', 'S7','S7'],  
 'Score': [85, 90, 88, 75, 78, 92, 95, 85, 90, 88, 75, 78]  
}  
  
df = pd.DataFrame(data)  
print("Original Data:\n", df)  
  
df['Mean by Class and section'] = df.groupby(['Class', 'section'])['Score'].transform(lambda x: x.mean())  
print("\nSD within each class:\n", df)  
**##### Contigency table**  
import pandas as pd  
  
# Sample categorical data  
data = {  
 'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female', 'Male', 'Female', 'Female', 'Male'],  
 'Class': ['A', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B'],  
 'Passed': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']  
}  
  
df = pd.DataFrame(data)  
print(" Sample Data:\n", df)  
  
contingency\_table = pd.crosstab([df['Class'], df['Gender']], df['Passed'], margins=True)  
  
print("\n Contingency Table (Class + Gender vs Passed):\n", contingency\_table)  
  
**#### Pivot table**  
import pandas as pd  
  
data = {  
 'Class': ['A', 'A', 'B', 'B', 'A', 'B'],  
 'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female'],  
 'Score': [85, 90, 88, 75, 78, 92]  
}  
df = pd.DataFrame(data)  
  
pivot = pd.pivot\_table(df, values='Score', index='Class', columns='Gender', aggfunc='sum')  
print("Pivot Table:\n", pivot)  
  
**##### Cross tab**  
crosstab = pd.crosstab(df['Gender'], df['Class'])  
print("\nCross-Tabulation:\n", crosstab)  
  
data2 = {  
 'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female'],  
 'Passed': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes']  
}  
df2 = pd.DataFrame(data2)  
  
contingency = pd.crosstab(df2['Gender'], df2['Passed'])  
print("\nContingency Table:\n", contingency)

**###### Chi\_square table**  
import pandas as pd  
from scipy.stats import chi2\_contingency  
  
# Data  
data2 = {  
 'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female','Female', 'Male', 'Male', 'Female'],  
 'Passed': ['Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No','No','Yes','No']  
}  
df2 = pd.DataFrame(data2)  
  
# Step 1: Create contingency table  
contingency = pd.crosstab(df2['Gender'], df2['Passed'])  
print("\n Contingency Table:\n", contingency)  
  
# Step 2: Apply Chi-Square test  
chi2, p, dof, expected = chi2\_contingency(contingency)  
  
# Step 3: Print results  
print("\n Chi-Square Test Results:")  
print(f"Chi-square statistic = {chi2:.4f}")  
print(f"Degrees of freedom = {dof}")  
print(f"P-value = {p:.4f}")  
print("\nExpected Frequencies:\n", pd.DataFrame(expected, index=contingency.index, columns=contingency.columns))  
  
  
row\_percent = contingency.div(contingency.sum(axis=1), axis=0) \* 100  
print("\nRow-wise Percentage (%):\n", row\_percent.round(2))  
  
# Assuming 'contingency' is your contingency table  
column\_percent = contingency.div(contingency.sum(axis=0), axis=1) \* 100  
  
# Print the column-wise percentage, rounded to 2 decimal places  
print("\nColumn-wise Percentage (%):\n", column\_percent.round(2))  
  
combined = contingency.astype(str) + " (" + row\_percent.round(1).astype(str) + "%" ")"  
print(" Contingency Table with Row-wise Percentages:\n", )  
print(combined)

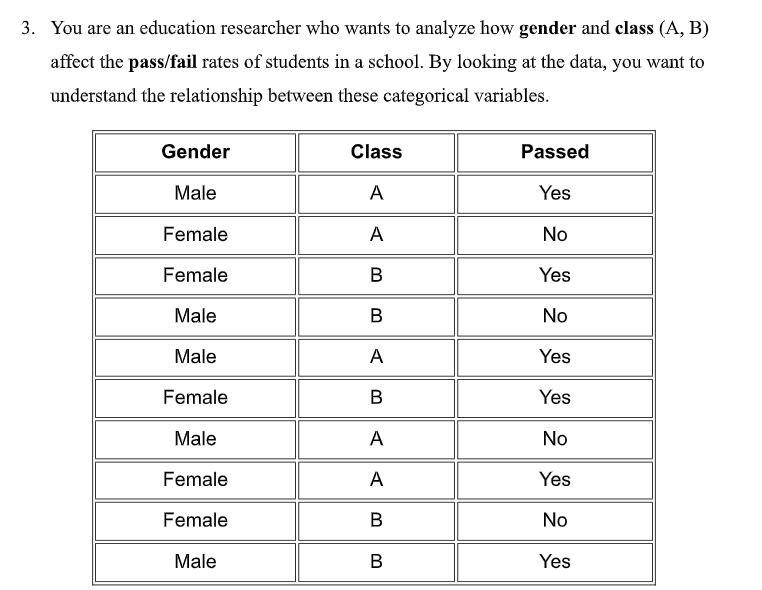
**Questions**

  
**#(i) Calculate the mean score for each class**  
import numpy as np  
  
# Scores  
class\_one = np.array([85, 90, 88])  
class\_three = np.array([75, 78])  
class\_five = np.array([92, 95])  
  
# Mean for each class  
mean\_one = np.mean(class\_one)  
mean\_three = np.mean(class\_three)  
mean\_five = np.mean(class\_five)  
  
print("Mean Scores:")  
print(f"Class One: {mean\_one:.2f}")  
print(f"Class Three: {mean\_three:.2f}")  
print(f"Class Five: {mean\_five:.2f}")  
  
**#(ii) Compare the average scores to determine the best performing class**  
# Compare means  
mean\_dict = {"Class One": mean\_one, "Class Three": mean\_three, "Class Five": mean\_five}  
best\_class = max(mean\_dict, key=mean\_dict.get)  
  
print("\nAverage Score Comparison:")  
for k, v in mean\_dict.items():  
 print(f"{k}: {v:.2f}")  
  
print(f"\nThe best performing class is: {best\_class}")  
  
**#(iii) Estimate standard deviation and coefficient of variation**  
# Standard deviation (sample SD) for each class  
std\_one = np.std(class\_one, ddof=1)  
std\_three = np.std(class\_three, ddof=1)  
std\_five = np.std(class\_five, ddof=1)  
  
# Coefficient of Variation (CV = SD / Mean \* 100)  
cv\_one = (std\_one / mean\_one) \* 100  
cv\_three = (std\_three / mean\_three) \* 100  
cv\_five = (std\_five / mean\_five) \* 100  
  
print("\nStandard Deviation and Coefficient of Variation:")  
print(f"Class One → SD = {std\_one:.2f}, CV = {cv\_one:.2f}%")  
print(f"Class Three → SD = {std\_three:.2f}, CV = {cv\_three:.2f}%")  
print(f"Class Five → SD = {std\_five:.2f}, CV = {cv\_five:.2f}%")  
  
**#(iv) Estimate mean score and standard deviation at a time (summary table)**  
import pandas as pd  
  
# Create summary DataFrame  
results = pd.DataFrame({  
 "Class": ["One", "Three", "Five"],  
 "Mean": [mean\_one, mean\_three, mean\_five],  
 "Standard Deviation": [std\_one, std\_three, std\_five],  
 "Coefficient of Variation (%)": [cv\_one, cv\_three, cv\_five]  
})  
  
print("\nSummary Table:\n")  
print(results.round(2))

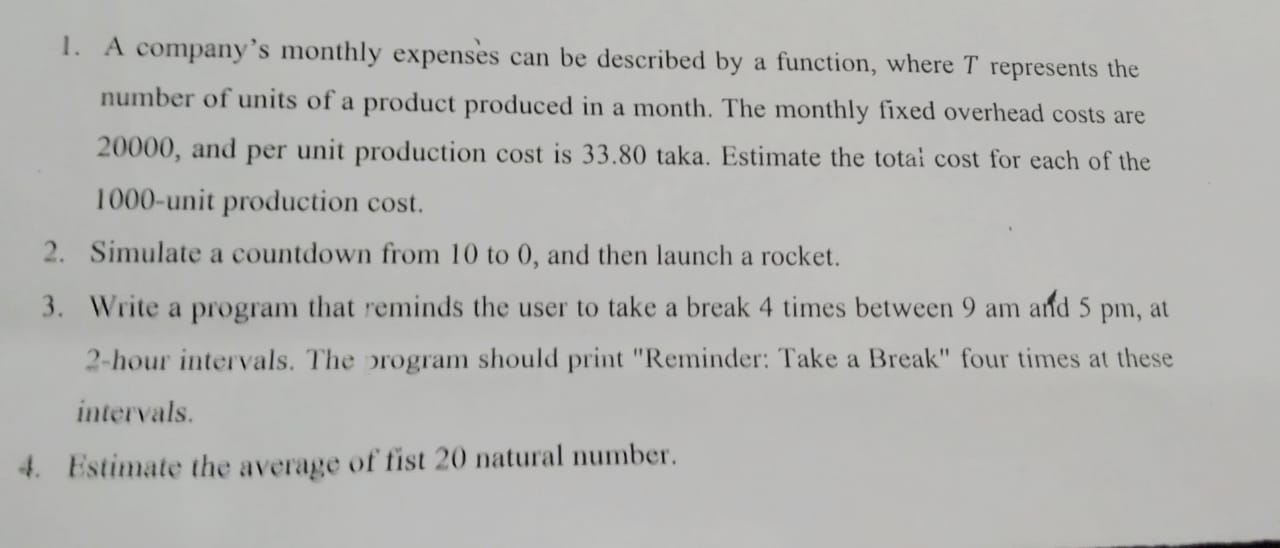


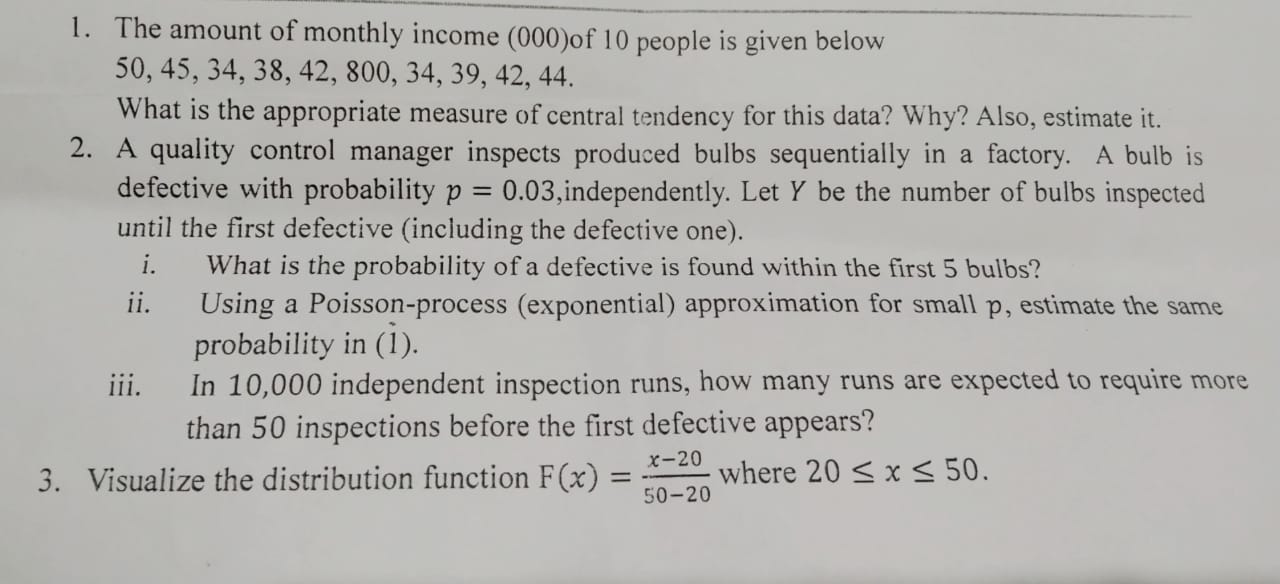
**#Organize the data**  
# Data: Scores of student sections in different classes  
classes = {  
 "Class A": {  
 "S1": [85, 88],  
 "S2": [90, 75],  
 "S4": [90],  
 "S5": [88]  
 },  
 "Class B": {  
 "S3": [78, 85],  
 "S4": [92],  
 "S5": [95]  
 },  
 "Class C": {  
 "S7": [75, 78]  
 }  
}

**#i. Calculate average score for each section**  
# Calculate average scores  
average\_scores = {}  
  
for class\_name, sections in classes.items():  
 average\_scores[class\_name] = {}  
 for section, scores in sections.items():  
 average\_scores[class\_name][section] = sum(scores) / len(scores)  
  
# Display results  
print("Average Scores for Each Section:")  
for class\_name, sections in average\_scores.items():  
 print(f"{class\_name}:")  
 for section, avg in sections.items():  
 print(f" {section}: {avg:.2f}")  
  
**#ii. Calculate standard deviation for each section**  
import math  
  
# Function to calculate standard deviation  
def std\_dev(scores):  
 mean = sum(scores) / len(scores)  
 variance = sum((x - mean) \*\* 2 for x in scores) / len(scores)  
 return math.sqrt(variance)  
  
# Calculate standard deviation  
std\_scores = {}  
  
for class\_name, sections in classes.items():  
 std\_scores[class\_name] = {}  
 for section, scores in sections.items():  
 std\_scores[class\_name][section] = std\_dev(scores)  
  
# Display results  
print("\nStandard Deviation for Each Section:")  
for class\_name, sections in std\_scores.items():  
 print(f"{class\_name}:")  
 for section, std in sections.items():  
 print(f" {section}: {std:.2f}")



**#Organize the data**  
import pandas as pd  
  
# Data  
data = {  
 'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female', 'Male', 'Female', 'Female', 'Male'],  
 'Class': ['A', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B'],  
 'Passed': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']  
}  
  
df = pd.DataFrame(data)  
print(df)  
  
**#Create contingency tables**  
# Contingency table: Gender vs Passed  
gender\_table = pd.crosstab(df['Gender'], df['Passed'])  
print("\nGender vs Passed:\n", gender\_table)  
  
# Contingency table: Class vs Passed  
class\_table = pd.crosstab(df['Class'], df['Passed'])  
print("\nClass vs Passed:\n", class\_table)  
  
**#Perform Chi-Square Test**  
from scipy.stats import chi2\_contingency  
  
# Chi-square test for Gender vs Passed  
chi2\_gender, p\_gender, dof\_gender, expected\_gender = chi2\_contingency(gender\_table)  
print(f"\nChi-square test (Gender vs Passed): chi2 = {chi2\_gender:.2f}, p-value = {p\_gender:.4f}")  
  
# Chi-square test for Class vs Passed  
chi2\_class, p\_class, dof\_class, expected\_class = chi2\_contingency(class\_table)  
print(f"Chi-square test (Class vs Passed): chi2 = {chi2\_class:.2f}, p-value = {p\_class:.4f}")

**CT Questions**  
**#1. Calculate total monthly cost**  
# Given data  
fixed\_overhead = 20000  
unit\_cost = 33.80  
total\_units = 1000  
  
# Calculate total cost for each unit from 1 to 1000  
for units\_produced in range(1, total\_units + 1):  
 total\_cost = fixed\_overhead + (unit\_cost \* units\_produced)  
 print(f"Total cost for {units\_produced} units: {total\_cost:.2f} taka")

**#2. Countdown from 10 to 0 and launch rocket**  
import time  
  
# Countdown  
for i in range(10, -1, -1):  
 print(i)  
 time.sleep(1) # Wait 1 second between counts (optional)  
  
print("Rocket Launched!")  
**#3.Reminder to take a break 4 times between 9 AM and 5 PM**  
# Break intervals: 9 AM, 11 AM, 1 PM, 3 PM  
break\_times = ["9:00 AM", "11:00 AM", "1:00 PM", "3:00 PM"]  
  
for time\_slot in break\_times:  
 print(f"Reminder at {time\_slot}: Take a Break")  
**#4. Average of first 20 natural numbers**  
# First 20 natural numbers  
numbers = list(range(1, 21))  
average = sum(numbers) / len(numbers)  
print(f"Average of first 20 natural numbers: {average}")  
  
  
**#1. Central Tendency for Monthly Income**  
import numpy as np  
  
# Monthly income data  
income = [50, 45, 34, 38, 42, 800, 34, 39, 42, 44]  
  
# Calculate mean, median, and mode  
mean\_income = np.mean(income)  
median\_income = np.median(income)  
  
print(f"Mean: {mean\_income}")  
print(f"Median: {median\_income} <-- Appropriate due to outlier")  
  
**#2. Geometric / Poisson Approximation for Bulbs**  
# Geometric: P(Y <= 5) = 1 - (1-p)^5  
prob\_within\_5 = 1 - (1 - p)\*\*n  
print(f"Probability first defective within 5 bulbs: {prob\_within\_5:.4f}")  
  
**#ii. Poisson-process (Exponential) approximation**  
import math  
  
prob\_approx = 1 - math.exp(-n \* p)  
print(f"Exponential approximation: {prob\_approx:.4f}")  
  
**#iii. Expected runs requiring more than 50 inspections**  
runs = 10000  
threshold = 50  
  
prob\_more\_50 = (1 - p)\*\*threshold  
expected\_runs = runs \* prob\_more\_50  
print(f"Expected runs needing >50 inspections: {expected\_runs:.0f}")  
  
**#3. Visualize the distribution function**  
import matplotlib.pyplot as plt  
import numpy as np  
  
# x values  
x = np.linspace(20, 50, 100)  
F\_x = (x - 20) / (50 - 20)  
  
# Plot  
plt.plot(x, F\_x, label='F(x)')  
plt.title('Distribution Function F(x)')  
plt.xlabel('x')  
plt.ylabel('F(x)')  
plt.grid(True)  
plt.legend()  
plt.show()