**Exp No.:1(A)**

**Date:04.09.25**

**Python Program to Calculate the Average of the Best Two Out of Three Internal Test Marks**

### **Aim:**

To write a Python program that reads the marks of three internal Mathematics tests for a student and calculates the average of the best two test scores.

### **Algorithm:**

1. **Start**
2. Prompt the user to **input marks** for Test 1, Test 2, and Test 3.
3. **Store** the three marks in a list.
4. **Sort** the list in descending order (highest marks first).
5. **Select** the first two marks (best two scores) from the sorted list.
6. **Calculate the average** of these two marks.
7. **Display** the average.
8. **End**

### 

### **Code:**

m1=float(input("Enter your marks in Internal test-I:"))

m2=float(input("Enter your marks in Internal test-II:"))

m3=float(input("Enter your marks in Internal test-III:"))

marks=[m1,m2,m3]

marks.sort(reverse=True)

best\_two=marks[:2]

avg=sum(best\_two)/2

print("The best two test marks:",best\_two)

print("Average of best two tests:",avg)

### 

### **Output:**

Enter your marks in Internal test-I:90

Enter your marks in Internal test-II:85

Enter your marks in Internal test-III:58

The best two test marks: [90.0, 85.0]

Average of best two tests: 87.5

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

### 

**Exp No.:1(B)**

**Date:04.09.25**

**Python Program to Calculate the Best Two-Test Average from Three Mini-Test Scores in a Coding Bootcamp**

### **Aim:**

To write a Python program that accepts decimal marks for three mini-tests taken by a bootcamp participant and calculates the average of the best two test scores.

### **Algorithm:**

1. **Start**
2. Prompt the user to **enter marks** for three tests (they may be decimal numbers).
3. **Store** the three marks in a list.
4. **Sort** the list in descending order using sort(reverse=True).
5. **Select the top two marks** (the first two in the sorted list).
6. **Calculate the average** of the top two marks.
7. **Display** the average rounded to two decimal places.
8. **End**

### **Code:**

test1 = float(input("Enter the marks for Test 1: "))

test2 = float(input("Enter the marks for Test 2: "))

test3 = float(input("Enter the marks for Test 3: "))

marks = [test1, test2, test3]

marks.sort(reverse=True)

average = (marks[0] + marks[1]) / 2

print(f"The average of the best two test marks is: {average:.2f}")

### **Output:**

Enter the marks for Test 1: 78.5

Enter the marks for Test 2: 91.0

Enter the marks for Test 3: 84.25

The average of the best two test marks is: 87.63

### **Result:**

Thus the program is successfully executed and the output is obtained and verified

**Exp No.:1(C)**

**Date: 04.09.2025**

# **Python Program to Calculate the Best Two-Trial Average from Three Athlete Performances**

### 

### **Aim:**

To write a Python program that records an athlete’s performance in three trials and calculates the average of the best two performances.

### 

### **Algorithm:**

1. **Start**
2. Prompt the user to enter the performance values for three trials (accept decimal input).
3. Store the values in a list.
4. Sort the list in descending order using sort(reverse=True).
5. Select the top two performances from the sorted list.
6. Calculate their average.
7. Display the average rounded to two decimal places.
8. **End**

### 

### **Program:**

trial1 = float(input("Enter performance for Trial 1: "))

trial2 = float(input("Enter performance for Trial 2: "))

trial3 = float(input("Enter performance for Trial 3: "))

performances = [trial1, trial2, trial3]

performances.sort(reverse=True)

average = (performances[0] + performances[1]) / 2

print(f"The average of the best two performances is: {average:.2f}")

### **Output:**

Enter performance for Trial 1: 12.5

Enter performance for Trial 2: 13.2

Enter performance for Trial 3: 11.8

The average of the best two performances is: 12.85

### **Result:**

Thus the program is successfully executed and the output is obtained and verified

**Exp No.:2(A)**

**Date:04.09.2025**

**Python Program to Check if a Transaction ID is a Palindrome and Count the Frequency of Each Digit**

### **Aim:**

To write a Python program that checks whether a given numeric transaction ID is a palindrome and also counts the frequency of each digit.

### **Algorithm:**

1. **Start**
2. Prompt the user to enter a transaction ID (as a number or string).
3. Convert the input to a string (to process digits easily).
4. Check if the transaction ID is a palindrome by comparing it with its reverse.
5. Initialize a dictionary (or list) to count digit frequencies.
6. Loop through each digit and update its count.
7. Display whether the ID is a palindrome.
8. Display the frequency of each digit.
9. **End**

### **Program:**

transaction\_id = input("Enter the transaction ID: ")

if transaction\_id == transaction\_id[::-1]:

print("The transaction ID is a palindrome.")

else:

print("The transaction ID is not a palindrome.")

digit\_freq = {}

for digit in transaction\_id:

if digit.isdigit():

digit\_freq[digit] = digit\_freq.get(digit, 0) + 1

print("Digit frequencies:")

for digit, count in sorted(digit\_freq.items()):

print(f"Digit {digit}: {count} time(s)")

### **Output:**

Enter the transaction ID: 12321

The transaction ID is a palindrome.

Digit frequencies:

Digit 1: 2 time(s)

Digit 2: 2 time(s)

Digit 3: 1 time(s)

### **Result:**

Thus the output is successfully executed and the output is obtained and verified.

**Exp No.:2(B)**

**Date:18.09.25**

**Python Program to Check if a Railway PNR Number is a Palindrome and Count the Frequency of Each Digit**

### **Aim:**

To write a Python program that verifies whether a given railway PNR number is a palindrome and prints the frequency of each digit in the number.

### **Algorithm:**

1. **Start**
2. Prompt the user to enter a railway PNR number (as input).
3. Convert the number to a string for easy digit processing.
4. Compare the PNR number with its reverse to check for a palindrome.
5. Initialize a dictionary to store digit frequencies.
6. Traverse each character in the PNR string:  
   * If it’s a digit, update its count in the dictionary.
7. Print whether the PNR number is a palindrome.
8. Print the frequency of each digit.
9. **End**

### **Program:**

pnr = input("Enter the railway PNR number: ")

if pnr == pnr[::-1]:

print("The PNR number is a palindrome.")

else:

print("The PNR number is not a palindrome.")

digit\_freq = {}

for char in pnr:

if char.isdigit():

digit\_freq[char] = digit\_freq.get(char, 0) + 1

print("Digit frequencies:")

for digit, count in sorted(digit\_freq.items()):

print(f"Digit {digit}: {count} time(s)")

### **Output:**

Enter the railway PNR number: 1234321

The PNR number is a palindrome.

Digit frequencies:

Digit 1: 2 time(s)

Digit 2: 2 time(s)

Digit 3: 2 time(s)

Digit 4: 1 time(s)

### **Result:**

Thus the program is successfully executed and the output is obtained and verified

**Exp No.:2(C)**

**Date:18.09.25**

**Python Program to Check if a School Roll Number is a Palindrome and Count the Frequency of Each Digit**

### **Aim:**

To write a Python program that checks whether a given school roll number is a palindrome and prints how many times each digit appears in the number.

### **Algorithm:**

1. **Start**
2. Prompt the user to enter a school roll number.
3. Convert the input to a string to allow reversal and digit analysis.
4. Compare the roll number with its reverse to check for palindrome.
5. Initialize an empty dictionary to track digit frequencies.
6. Loop through each character in the roll number:  
   * If it's a digit, increment its count in the dictionary.
7. Display whether the roll number is a palindrome.
8. Display how many times each digit appears.
9. **End**

### **Program:**

roll\_number = input("Enter the school roll number: ")

if roll\_number == roll\_number[::-1]:

print("The roll number is a palindrome.")

else:

print("The roll number is not a palindrome.")

digit\_freq = {}

for char in roll\_number:

if char.isdigit():

digit\_freq[char] = digit\_freq.get(char, 0) + 1

print("Digit frequencies:")

for digit, count in sorted(digit\_freq.items()):

print(f"Digit {digit}: {count} time(s)")

### **Output:**

Enter the school roll number: 1221

The roll number is a palindrome.

Digit frequencies:

Digit 1: 2 time(s)

Digit 2: 2 time(s)

### **Result:**

Thus the program is successfully executed and the output is obtained and verified

**Exp No.:3(A)**

**Date:09.10.25**

**Fibonacci Series**

### **Aim:**

To write a Python program to display the Fibonacci series up to N terms, where N is entered by the user.

### **Algorithm:**

1. **Start** the program.
2. **Input** the number of terms N from the user.
3. **Check** if N is greater than 0.  
   * If not, display an error message and stop.
4. **Initialize** the first two terms: a = 0, b = 1.
5. **Display** a message indicating the start of the Fibonacci series.
6. **Repeat** the following steps N times:  
   * Print the current term a.
   * Update the values: a, b = b, a + b.
7. **End** the program.

### **Program:**

# Program to display Fibonacci series up to N terms

# Step 1: Input number of terms

N = int(input("Enter the number of terms (N > 0): "))

# Step 2: Check if N is valid

if N <= 0:

print("Please enter a number greater than 0.")

else:

# Step 3: Initialize first two terms

a, b = 0, 1

print("Fibonacci Series:")

# Step 4: Loop to generate series

for i in range(N):

print(a, end=" ")

a, b = b, a + b

### **Output:**

Enter the number of terms (N > 0): 7

Fibonacci Series:

0 1 1 2 3 5 8

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:3(B)**

**Date:09.10.25**

**Seat Arrangement using Fibonacci Series**

### **Aim:**

To write a Python program that displays a Fibonacci-style seating arrangement up to N seats, where N is entered by the user.

### **Algorithm:**

1. **Start** the program.
2. **Input** the number of seats N from the user.
3. **Check** if N is greater than 0.  
   * If not, display an error message and stop.
4. **Initialize** the first two seat numbers: a = 0, b = 1.
5. **Display** a heading: "Fibonacci Seating Arrangement:".
6. **Repeat** the following steps N times:  
   * Print the current seat number a.
   * Update the values using: a, b = b, a + b.
7. **End** the program.

**Program:**

# Program to display Fibonacci-style seating arrangement

# Step 1: Input number of seats

N = int(input("Enter the number of seats (N > 0): "))

# Step 2: Validate input

if N <= 0:

print("Please enter a number greater than 0.")

else:

# Step 3: Initialize first two seat counts

a, b = 0, 1

print("Fibonacci Seating Arrangement:")

# Step 4: Generate and display Fibonacci sequence

for i in range(N):

print(a, end=" ")

a, b = b, a + b

### **Output:**

Enter the number of seats (N > 0): 6

Fibonacci Seating Arrangement:

0 1 1 2 3 5

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:3(B)**

**Date:09.10.25**

**Garden Layout using Fibonacci Series**

### **Aim:**

To write a Python program that generates a Fibonacci sequence representing the layout of garden sections, up to N terms specified by the user.

### **Algorithm:**

1. **Start** the program.
2. **Input** the number of garden sections N from the user.
3. **Check** if N is greater than 0.  
   * If not, display an error message and stop.
4. **Initialize** the first two Fibonacci numbers: a = 0, b = 1.
5. **Display** a heading: "Fibonacci Garden Layout:".
6. **Repeat** the following steps N times:  
   * Print the current Fibonacci number a.
   * Update the values using a, b = b, a + b.
7. **End** the program.

**Program:**

# Program to generate Fibonacci sequence for garden layout

# Step 1: Input number of terms

N = int(input("Enter the number of garden sections (N > 0): "))

# Step 2: Validate input

if N <= 0:

print("Please enter a number greater than 0.")

else:

# Step 3: Initialize first two Fibonacci numbers

a, b = 0, 1

print("Fibonacci Garden Layout:")

# Step 4: Generate Fibonacci sequence

for i in range(N):

print(a, end=" ")

a, b = b, a + b

### **Output:**

Enter the number of garden sections (N > 0): 7

Fibonacci Garden Layout:

0 1 1 2 3 5 8

### 

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:4(A)**

**Date:30.10.25**

**Binary Product Code to Decimal Converter**

### **Aim:**

To write a Python program that converts a **binary product code** (a number containing only 0s and 1s) into its **decimal equivalent**.

### **Algorithm:**

1. **Start** the program.
2. **Input** the binary product code as a string from the user.
3. **Check** if the input contains only '0' and '1' characters.  
   * If not, display an error message and stop.
4. **Use** the built-in function int(binary\_code, 2) to convert the binary number into a decimal value.
5. **Store** the result in the variable decimal\_value.
6. **Display** the decimal equivalent of the entered binary code.
7. **End** the program.

**Program:**

# Program to convert binary product code to decimal

# Step 1: Input binary product code

binary\_code = input("Enter the binary product code: ")

# Step 2: Validate input

if not all(bit in '01' for bit in binary\_code):

print("Invalid binary number! Please enter only 0s and 1s.")

else:

# Step 3: Convert binary to decimal

decimal\_value = int(binary\_code, 2)

# Step 4: Display result

print("The decimal equivalent of the product code is:", decimal\_value)

### **Output:**

Enter the binary product code: 1011

The decimal equivalent of the product code is: 11

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:4(B)**

**Date:30.10.25**

**Octal to Hexadecimal Converter**

### **Aim:**

To write a Python program that converts an octal reading (base 8) into its hexadecimal equivalent (base 16).

### **Algorithm:**

1. **Start** the program.
2. **Input** the octal reading as a string from the user.
3. **Validate** that the input contains only digits from 0 to 7.  
   * If invalid, display an error message and stop.
4. **Convert** the valid octal number to its **decimal equivalent** using:  
    decimal\_value = int(octal\_value, 8)
5. **Convert** the decimal value to **hexadecimal** using:  
    hexadecimal\_value = hex(decimal\_value)[2:].upper()
6. **Display** the hexadecimal equivalent of the given octal number.
7. **End** the program.

**Program:**

# Program to convert octal readings to hexadecimal

# Step 1: Input octal reading

octal\_value = input("Enter the octal reading: ")

# Step 2: Validate input (only digits 0-7)

if not all(ch in '01234567' for ch in octal\_value):

print("Invalid octal number! Please enter digits 0–7 only.")

else:

# Step 3: Convert octal to decimal

decimal\_value = int(octal\_value, 8)

# Step 4: Convert decimal to hexadecimal

hexadecimal\_value = hex(decimal\_value)[2:].upper()

# Step 5: Display the result

print("Hexadecimal equivalent:", hexadecimal\_value)

**Output:**

Enter the octal reading: 157

Hexadecimal equivalent: 6F

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:4(C)**

**Date:30.10.25**

**Binary Access Key to Decimal and Validity Verification**

### **Aim:**

To write a Python program that converts a **binary access key** into its **decimal equivalent** and verifies whether the key is **valid or invalid** based on a set threshold.

### **Algorithm:**

1. **Start** the program.
2. **Input** the binary access key from the user.
3. **Check** if the key contains only '0' and '1'.  
   * If not, display an error message and stop.
4. **Convert** the binary key to its decimal equivalent using:  
    decimal\_key = int(binary\_key, 2)
5. **Display** the decimal equivalent of the key.
6. **Verify validity:**
   * If decimal\_key > 100, display **“Access Granted ”**.
   * Otherwise, display **“Access Denied ”**.
7. **End** the program.

**Program:**

# Program to convert a binary access key to decimal and verify its validity

# Step 1: Input binary key

binary\_key = input("Enter the binary access key: ")

# Step 2: Validate that input has only 0s and 1s

if not all(bit in '01' for bit in binary\_key):

print("Invalid access key! Please enter only binary digits (0 or 1).")

else:

# Step 3: Convert binary to decimal

decimal\_key = int(binary\_key, 2)

# Step 4: Display the decimal equivalent

print("Decimal equivalent of access key:", decimal\_key)

# Step 5: Verify validity

if decimal\_key > 100:

print("Access Granted !")

else:

print("Access Denied !")

**Output:**

Enter the binary access key: 1100101

Decimal equivalent of access key: 101

Access Granted!

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:5(A)**

**Date:30.10.25**

# **Chat Message Analysis**

### **Aim:**

To write a Python program that analyzes a chat message and counts the **number of words, digits, uppercase letters, and lowercase letters** in the message.

### **Algorithm:**

1. **Start** the program.
2. **Input** a chat message from the user.
3. **Initialize counters**:  
   * words for total words (using len(message.split()))
   * digits, upper, and lower as 0.
4. **Loop** through each character in the message:  
   * If the character is a digit → increment digits.
   * If it is an uppercase letter → increment upper.
   * If it is a lowercase letter → increment lower.
5. **After the loop**, all counts will be ready.
6. **Display** the counts of words, digits, uppercase, and lowercase letters.
7. **End** the program.

**Program:**

# Program to count words, digits, uppercase and lowercase letters in a chat message

# Step 1: Input message from user

message = input("Enter the chat message: ")

# Step 2: Initialize counters

words = len(message.split())

digits = 0

upper = 0

lower = 0

# Step 3: Loop through each character

for ch in message:

if ch.isdigit():

digits += 1

elif ch.isupper():

upper += 1

elif ch.islower():

lower += 1

# Step 4: Display the results

print("\nChat Log Analysis:")

print("Words:", words)

print("Digits:", digits)

print("Uppercase Letters:", upper)

print("Lowercase Letters:", lower)

**Output:**

Enter the chat message: Hello World 2025!

Chat Log Analysis:

Words: 3

Digits: 4

Uppercase Letters: 2

Lowercase Letters: 8

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:5(B)**

**Date:30.10.25**

**Feedback Analysis**

### **Aim:**

To write a Python program that analyzes a **teacher’s feedback** and counts the **number of words, numbers, uppercase letters, and lowercase letters** present in the feedback.

### **Algorithm (7 Steps):**

1. **Start** the program.
2. **Input** the feedback text from the teacher.
3. **Initialize counters** for counting:  
   * words → total number of words using len(feedback.split()).
   * numbers, uppercase, and lowercase → all set to 0.
4. **Loop** through each character in the feedback:  
   * If the character is a digit → increment numbers.
   * If it is an uppercase letter → increment uppercase.
   * If it is a lowercase letter → increment lowercase.
5. **After the loop**, all the counts will be calculated.
6. **Display** the total number of words, numbers, uppercase, and lowercase letters.
7. **End** the program.

**Program:**

# Program to count words, numbers, uppercase and lowercase letters in feedback

# Step 1: Input feedback from teacher

feedback = input("Enter the student feedback: ")

# Step 2: Initialize counters

words = len(feedback.split())

numbers = 0

uppercase = 0

lowercase = 0

# Step 3: Loop through each character

for ch in feedback:

if ch.isdigit():

numbers += 1

elif ch.isupper():

uppercase += 1

elif ch.islower():

lowercase += 1

# Step 4: Display results

print("\nFeedback Analysis:")

print("Words:", words)

print("Numbers:", numbers)

print("Uppercase Letters:", uppercase)

print("Lowercase Letters:", lowercase)

**Output:**

Enter the student feedback: Excellent Work! You scored 95 marks.

Feedback Analysis:

Words: 6

Numbers: 2

Uppercase Letters: 2

Lowercase Letters: 24

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.

**Exp No.:5(B)**

**Date:30.10.25**

**Newspaper Headline Analysis**

### **Aim:**

To write a Python program that analyzes a **newspaper headline** and counts the **number of words, uppercase letters, lowercase letters, digits, and special characters**.

### **Algorithm (7 Steps):**

1. **Start** the program.
2. **Input** the newspaper headline from the user.
3. **Initialize counters**:  
   * words → total words using len(headline.split())
   * uppercase, lowercase, digits, special → all set to 0
4. **Loop** through each character in the headline:  
   * If the character is uppercase → increment uppercase
   * If lowercase → increment lowercase
   * If digit → increment digits
   * If neither letter, digit, nor space → increment special
5. **After the loop**, all counts are ready.
6. **Display** the statistical breakdown of the headline.
7. **End** the program.

**Program:**

# Program to analyze a newspaper headline

headline = input("Enter a newspaper headline: ")

words = len(headline.split())

uppercase = 0

lowercase = 0

digits = 0

special = 0

for ch in headline:

if ch.isupper():

uppercase += 1

elif ch.islower():

lowercase += 1

elif ch.isdigit():

digits += 1

elif not ch.isspace():

special += 1

print("\nHeadline Statistical Breakdown:")

print("Words:", words)

print("Uppercase Letters:", uppercase)

print("Lowercase Letters:", lowercase)

print("Digits:", digits)

print("Special Characters:", special)

**Output:**

Enter a newspaper headline: Hello World!

Headline Statistical Breakdown:

Words: 2

Uppercase Letters: 2

Lowercase Letters: 8

Digits: 0

Special Characters: 1

### **Result:**

Thus the program is successfully executed and the output is obtained and verified.