S.No: 1	Exp. Name: ***Eldest of Three***	**Date: 2025-07-15**

Aim:

In a classroom, a teacher wants to identify the eldest student among the three. To simplify this task, the teacher decides to write a Python program that takes the ages of the three students and determines the oldest.

Can you help the teacher by writing this program?

Input Format:

- Three integers are entered on separate lines, each representing the age of a student.

Output Format:

- The output is the integer that represents the age of the elder student.

Source Code:

eldestAge.py

```
age1=int(input())
age2=int(input())
age3=int(input())
print(max(age1,age2,age3))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
12
14
13
14

Test Case - 2
User Output
11

S.No: 2	Exp. Name: ***Prime numbers from 2 to the given positive number***	**Date: 2025-07-22**

Aim:

Implement a program that takes an integer n input from the user and prints all the prime numbers from 2 to n.

Note: A prime number is divisible only by 1 and itself.

Input format:

- The input contains a positive integer n.

Output Format:

- Contains the list of prime numbers up to n

Example:

Input:

10

Output:

[2, 3, 5, 7]

Constraints:

- The input n should be a positive integer greater than or equal to 2.
- The algorithm should be efficient, aiming for a time complexity of approximately $O(n * sqrt(n))$ where n is the input number.
- The algorithm should handle large inputs reasonably well, without consuming excessive memory or taking an unreasonable amount of time to execute.

Source Code:

CTP32252.py

```
y=int(input())
pn=[]
for i in range(2,y+1):
        j=2
        flag=0
        while(j<(i//2)+1):
                if(i%j==0):
                        flag=1
                        break
                j=j+1
        if(flag==0):
                pn.append(i)
print(pn)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
10
[2, 3, 5, 7]

Test Case - 2
User Output
20
[2, 3, 5, 7, 11, 13, 17, 19]

Test Case - 3
User Output
30
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

S.No: 3	Exp. Name: ***Swap two numbers***	**Date: 2025-07-15**

Aim:

Write a Python program that reads two numbers from the user and swaps their values using only two variables.

Source Code:

swapping.py

```
a=float(input("Enter the first number: "))
b=float(input("Enter the second number: "))
print("Original numbers:")
print("First number:",a)
print("Second number:",b)
a=a+b
b=a-b
a=a-b
print("Swapped numbers:")
print("First number:",a)
print("Second number:",b)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the first number:
2
Enter the second number:
3
Original numbers:
First number: 2.0
Second number: 3.0
Swapped numbers:
First number: 3.0
Second number: 2.0

Test Case - 2

User Output
Enter the first number:
-3.5
Enter the second number:
2.5
Original numbers:
First number: -3.5
Second number: 2.5
Swapped numbers:
First number: 2.5
Second number: -3.5

S.No: 4	Exp. Name: ***Arithmetic Operations (Addition, Subtraction, Multiplication, Division)***	**Date: 2025-07-15**

Aim:

Write a program to read two integer inputs from the user and perform the following arithmetic operations: **addition, subtraction, multiplication,** and **division,** and print the result.

Input Format:

- The first two lines represent the input of two integers $num1$ and $num2$.

Output Format:

- The first line should print the sum of the provided inputs.
- The second line should print the subtraction of the provided inputs.
- The third line should print the multiplication of the provided inputs.
- The fourth line should print the division of the provided inputs.

Sample Test Case:

Input:

```
num1: 40
num2: 20
```

Output:

```
Addition of 40 and 20 = 60
Subtraction of 40 and 20 = 20
Multiplication of 40 and 20 = 800
Division of 40 and 20 = 2.0
```

Source Code:

Arithexample2.py

```
a=int(input("num1: "))
b=int(input("num2: "))
print(f"Addition of {a} and {b} = {a+b}")
print(f"Subtraction of {a} and {b} = {a-b}")
print(f"Multiplication of {a} and {b} = {a*b}")
print(f"Division of {a} and {b} = {a/b}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

num1:
40
num2:
20
Addition of 40 and 20 = 60
Subtraction of 40 and 20 = 20
Multiplication of 40 and 20 = 800
Division of 40 and 20 = 2.0

Test Case - 2
User Output
num1:
4
num2:
6
Addition of 4 and 6 = 10
Subtraction of 4 and 6 = -2
Multiplication of 4 and 6 = 24
Division of 4 and 6 = 0.6666666666666666

S.No: 5	Exp. Name: ***Logical Operators***	Date: 2025-07-15

Aim:

Write a Python program that demonstrates the usage of logical operators. Your program should read two integer values from the user and perform logical operations (`and, or, not`). Display the corresponding output for each operation.

Input Format:

- The program will prompt the user to input two integer values: $num1$ and $num2$. Each input will be provided on a separate line.

Output Format:

- Print the result of $num1$ and $num2$ in the format "<num1> and <num2>: <result>"
- Print the result of $num1$ or $num2$ in the format "<num1> or <num2>: <result>"
- Print the result of not $num1$ in the format "not <num1>: <result>"
- Print the result of not $num2$ in the format "not <num2>: <result>"

Source Code:

logicoperator.py

```
a=int(input())
b=int(input())
print(f"{a} and {b}: {a and b}")
print(f"{a} or {b}: {a or b}")
print(f"not {a}: {not a}")
print(f"not {b}: {not b}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
4
3 and 4: 4
3 or 4: 3
not 3: False
not 4: False

Test Case - 2
User Output
0
1
0 and 1: 0
0 or 1: 1
not 0: True
not 1: False

Test Case - 3
User Output
0
0
0 and 0: 0
0 or 0: 0
not 0: True
not 0: True

S.No: 6	Exp. Name: ***Comparison Operators***	Date: 2025-07-15

Aim:

Write a Python program that reads two integers from the user and demonstrates the usage of comparison operators. The program should perform all six comparison operations (==, !=, <, <=, >, >=) and print the result of each operation.

Input Format:

- The program will read two integers $num1, num2$, each on a new line.

Output format:

- For each comparison, print the comparison operation along with the result (True or False), formatted as follows:

```
{num1} (comparison operation) {num2} : {boolean result}
```

Source Code:

operators.py

```
num1=int(input())
num2=(int(input()))


print(f"{num1} == {num2} : {num1 == num2}")
print(f"{num1} != {num2} : {num1 != num2}")
print(f"{num1} < {num2}  : {num1 < num2}")
print(f"{num1} <= {num2} : {num1 <= num2}")
print(f"{num1} > {num2}  : {num1 > num2}")
print(f"{num1} >= {num2} : {num1 >= num2}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
4
4 == 4 : True
4 != 4 : False
4 < 4 : False
4 <= 4 : True

```
4 >= 4 : True
```

Test Case - 2
User Output
3
2
3 == 2 : False
3 != 2 : True
3 < 2 : False
3 <= 2 : False
3 > 2 : True
3 >= 2 : True

Test Case - 3
User Output
5
5
5 == 5 : True
5 != 5 : False
5 < 5 : False
5 <= 5 : True
5 > 5 : False
5 >= 5 : True

S.No: 7	Exp. Name: ***Writing an example using identity operator "is not"***	**Date: 2025-07-15**

Aim:

Take two integers `x` and `y` as input from the console using `input()` function. Using the identity operator `is not` check if `x` and `y` are same objects or not, print to the console, the result of the two input integers as shown in the example.

Sample Input and Output 1:

```
x: -8
y: -8
-8 is not -8 True
```

Sample Input and Output 2:

```
x: 256
y: 256
256 is not 256 False
```

Source Code:

IdenopExample3.py

```
# Take user input for x
x=int(input("x: "))
# Take user input for y
y=int(input("y: "))
# check whether x is not y and print the result
print(x, 'is not', y,(x is not y))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
x:
-8
y:
-8

```
-8 is not -8 True
```

Test Case - 2
User Output
x:
256
y:
256
256 is not 256 False

S.No: 8	Exp. Name: ***Bitwise Operator***	Date: 2025-07-15

Aim:

Take two integers x and y as inputs from the console using input() function. For each bitwise operator (>> , <<, &, |, ~, and ^), print to the console, the result of applying these operators on the two input integers as shown in the example.

Sample Input and Output:

```
x: 52
y: 20
52 >> 20 is 0
52 << 20 is 54525952
52 & 20 is 20
52 | 20 is 52
~ 52 is -53
52 ^ 20 is 32
```

Source Code:

Bitopexample1.py

```
#Type Code here...
x=int(input("x: "))
y=int(input("y: "))
print(f"{x} >> {y} is {x >> y}")
print(f"{x} << {y} is {x << y}")
print(f"{x} & {y} is {x & y}")
print(f"{x} | {y} is {x | y}")
print(f"~ {x} is {~x}")
print(f"{x} ^ {y} is {x^y}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
x:
30
y:
2
30 >> 2 is 7
30 << 2 is 120

30 & 2 is 2
30 | 2 is 30
~ 30 is -31
30 ^ 2 is 28

Test Case - 2
User Output
x:
10
y:
20
10 >> 20 is 0
10 << 20 is 10485760
10 & 20 is 0
10 | 20 is 30
~ 10 is -11
10 ^ 20 is 30

S.No: 9	Exp. Name: ***Write a program on assignment operators =,+=,-= and *=***	**Date: 2025-07-15**

Aim:

Take two integers `x` and `y` as inputs from the console using `input()` function. For each assignment operator (`=` , `+=`, `-=`, and `*=`), print to the console, the result of applying these operators on the two input integers as shown in the example.

Assumption:

- y is a non-zero integer

Sample Input and Output:

```
x: 20
y: 30
x = y: 30
x += y: 50
x -= y: -10
x *= y: 600
```

Hint: Before every assignment operation, reset the values of x and y to the initial values.

Source Code:

Assignexample2.py

```
# Assignment Operators =, +=, -=, *=
x=int(input("x: "))
y=int(input("y: "))
print(f"x = y: {y}")
print(f"x += y: {x+y}")
print(f"x -= y: {x-y}")
print(f"x *= y: {x*y}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
x:
20
y:
30

x = y: 30
x += y: 50
x -= y: -10
x *= y: 600

Test Case - 2
User Output
x:
6
y:
3
x = y: 3
x += y: 9
x -= y: 3
x *= y: 18

S.No: 10	Exp. Name: ***Ternary Operator***	Date: 2025-07-15

Aim:

Write a python program to find the maximum value of given two numbers using ternary operator.

Input Format:

- First line should prompt the user to enter first integer
- Second line should prompt the user to enter second integer

Output Format:

- Output should print the maximum value using ternary operator

Source Code:

ternary.py

```
x = int(input())
y = int(input())
max_value = x if x>y else y
print(max_value)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
91
123
123

Test Case - 2
User Output
15
-23
15

S.No: 11	Exp. Name: ***Membership Operator***	Date: 2025-07-15

Aim:

Write a Python program to demonstrate the membership operator.

A predefined list of integers is already provided to you.

Input Format:

- The program should prompt the user to enter an integer number.

Output Format:

- Use the `in` operator to check whether the given integer is present in the list or not and print the result.
- Use the `not in` operator to check whether the given integer is not present in the list and print the result.

Source Code:

membership.py

```
num_list = [1, 2, 3, 4, 5, 15, 25, 89, 54, 123, 489]
# Write your code here...
a=int(input())
print(a in num_list)
print(a not in num_list)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
21
False
True

Test Case - 2
User Output

-3
False
True

Test Case - 3
User Output
5
True
False

Test Case - 4
User Output
54
True
False

S.No: 12	Exp. Name: ***Write a program to addition, subtraction and multiplication and division of two complex numbers.***	**Date: 2025-07-15**

Aim:

Take **two complex numbers** as input from the console. For each arithmetic operator(+, -, *, and /), perform these operations on the two complex numbers and print the result as shown in the example.

Sample Input and Output:

```
a1: 10+3j
b2: 14+3j
a1 + b2 = (24+6j)
a1 - b2 = (-4+0j)
a1 * b2 = (131+72j)
a1 / b2 = (0.7268292682926829+0.05853658536585366j)
```

Source Code:

ComplexNums.py

```
# Take input for the first complex number c1
c1=complex(input("a1: "))
# Take input for the second complex number c2
c2=complex(input("b2: "))
# perform + operation on the complex numbers and print the result
print("a1 + b2 =", (c1 + c2))

# subtract c2 from c1 and print the result
print("a1 - b2 =", (c1 - c2))

# multiple c1, c2 and print the result
print("a1 * b2 =", (c1 * c2))

# divide c2 from c1 and print the result
print("a1 / b2 =", (c1 /c2))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
a1:

10+3j
b2:
41+3j
a1 + b2 = (51+6j)
a1 - b2 = (-31+0j)
a1 * b2 = (401+153j)
a1 / b2 = (0.24792899408284022+0.055029585798816574j)

Test Case - 2
User Output
a1:
45+9j
b2:
15+8j
a1 + b2 = (60+17j)
a1 - b2 = (30+1j)
a1 * b2 = (603+495j)
a1 / b2 = (2.58477508650519-0.7785467128027682j)

S.No: 13	Exp. Name: ***Print Multiplication Table***	Date: 2025-07-22

Aim:

Write a Python program to print the multiplication table of a given number in the given format.

Input Format:

- Prompt the user to enter an integer n for which they want to generate the multiplication table with the statement "Enter a number : ".

Output Format:

- Print the multiplication table of the entered number in the following format for numbers from 1 to 10.
- Each line should be in the format:

`{n}*{i}={result}`

where$i$ ranges from 1 to 10, and$result = n \times i$.

Note:

- Refer to visible test cases for better understanding.

Source Code:

multiplication_table.py

```
# Type Content here...
n=int(input("Enter a number : "))
for i in range(1,11):
        result=n*i
        print(f"{n}*{i}={result}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a number :
2
2*1=2
2*2=4
2*3=6
2*4=8

2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20

Test Case - 2
User Output
Enter a number :
13
13*1=13
13*2=26
13*3=39
13*4=52
13*5=65
13*6=78
13*7=91
13*8=104
13*9=117
13*10=130

Test Case - 3
User Output
Enter a number :
-10
-10*1=-10
-10*2=-20
-10*3=-30
-10*4=-40
-10*5=-50
-10*6=-60
-10*7=-70
-10*8=-80
-10*9=-90
-10*10=-100

S.No: 18	Exp. Name: ***Function with Multiple Return Values***	**Date: 2025-07-22**

Aim:

Write a Python program that defines a function with multiple return values to calculate the sum, mean, and maximum of the given list of numbers.

Input Format:

- A list of space-separated integers.

Output Format:

- The total sum of the integers.
- The mean (average) of the integers, rounded to two decimal places.
- The maximum value among the integers.

Constraints:

- $0 < length\ of\ the\ list <= 100$

Source Code:

multiplefun.py

```
def calculate_stats(n):
        total=0
        maximum=n[0]
        for num in n:
                total=total+num
                if num > maximum:
                        maximum=num
        mean = total/len(n)
        return total,mean,maximum




numbers = list(map(int, input().split()))
total, mean, maximum = calculate_stats(numbers)
print(total)
print(round(mean,2))
print(maximum)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
12 23 34 45 56 67 78 89
404
50.5
89

Test Case - 2
User Output
-23 -45 -56 -90 -12 -87
-313
-52.17
-12

S.No: 19	Exp. Name: ***Default Arguments Function***	Date: 2025-08-05

Aim:

Write a program to define a function using default arguments.

Input Format:

- First line of input should prompt the user to enter the name by displaying "-1 for default: ", If -1 is entered, the default value "Dude" is used.
- Second line of input should prompt the user to enter the age by displaying: "-1 for default: ", If -1 is entered, the default value "0" is used.
- Third line of input should prompt the user to enter the country by displaying: "-1 for default: ", If -1 is entered, the default value "Space" is used.

Output Format: The output should print the following line with provided inputs.

- Hello, {name}! You are {age} years old and you are from {country}

Source Code:

default.py

```
# Define a function with default arguments
def greet(name="Dude" , age="0", country="Space"):
        print(f"Hello, {name}! You are {age} years old and you are
from {country}")


# Get user input function
def get_user_input():
        name_input = input("-1 for default: ").strip()
        age_input = input("-1 for default: ").strip()
        country_input = input("-1 for default: ").strip()

        name = name_input if name_input != "-1" else "Dude"
        age = age_input if age_input != "-1" else "0"
        country = country_input if country_input != "-1" else "Space"

        return name, age, country

name, age, country = get_user_input()
greet(name, age, country)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
-1 for default:
-1
-1 for default:
-1
-1 for default:
-1
Hello, Dude! You are 0 years old and you are from Space

Test Case - 2
User Output
-1 for default:
Ram
-1 for default:
-1
-1 for default:
-1
Hello, Ram! You are 0 years old and you are from Space

Test Case - 3
User Output
-1 for default:
Vikas
-1 for default:
26
-1 for default:
America
Hello, Vikas! You are 26 years old and you are from America

S.No: 20	Exp. Name: ***Length of the String***	Date: 2025-07-22

Aim:

Write a program to find the length of the string without using any libraryfunctions.

Input Format:

A single line containing a string s.

Output Format:

A single integer representing the length of the input string.

Source Code:

length.py

```
def string_length(s):
        len=0
        for i in s:
                len=len+1
        return len



input_string = input() # Taking user input
length = string_length(input_string) # Calculating length using the
function
print(length) # Printing the length
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Code Tantra
11

Test Case - 2
User Output
Hello@World!!
13

Test Case - 3
User Output
12345678
8

S.No: 21	Exp. Name: ***Count of Substring in given string***	**Date: 2025-08-05**

Aim:

Write a program to count the number of occurrences of a given substring within a string, reading the string from left to right.

Constraints:

- 1 <= len(string) <=200.
- Each character in the string is an ASCII character.

Input Format:

- The first line of input contains the main string.
- The second line contains the substring.

Output Format:

- Output is an integer representing the total number of occurrences of the substring in the main string.

Source Code:

SubstringCount.py

```
main_string = input().strip()
sub_string = input().strip()
count = 0
start = 0
while True:
        start = main_string.find(sub_string, start)
        if start == -1:
                break
        count += 1
        start += 1
print(count)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
ABCDCDC
CDC
2

Test Case - 2
User Output
CodeTantra
co
0

S.No: 22	Exp. Name: ***List operations***	**Date: 2025-07-22**

Aim:

Write a program to perform the following operations on a list.
i. Addition
ii. Insertion
iii. Slicing

Input Format:

- The first line should take space-separated integers to form the list.
- The second line of input should prompt the user to enter two integers separated by a space representing the key and value for the insertion.
- The third line should take space-separated start index and end index for slicing.

Output Format:

- First line should display the initial list.
- Second line should display the list after insertion.
- Third line should display the list after slicing.

Example 1:

Input:

1 2 3 4 5
0 5
0 3

Output

[1, 2, 3, 4, 5]
[5, 1, 2, 3, 4, 5]
[5, 1, 2]

Note:

- In Python, adding an element with append adds it to the end of a list. To insert an element at a specific index, insert operation need to be used.

Source Code:

listOp.py

```
data = input()
num_list=[int(x) for x in data.split()]

key_value=input()
key, value=[int(x) for x in key_value.split()]

slice_input=input()
start, end=[int(x) for x in slice_input.split()]

print(num_list)

num_list.insert(key, value)
print(num_list)

print(num_list[start:end])
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1 2 3 4 5
0 5
0 3
[1, 2, 3, 4, 5]
[5, 1, 2, 3, 4, 5]
[5, 1, 2]

Test Case - 2
User Output
12 52 65 41 23 89 654
7 485
0 8
[12, 52, 65, 41, 23, 89, 654]
[12, 52, 65, 41, 23, 89, 654, 485]
[12, 52, 65, 41, 23, 89, 654, 485]

S.No: 23	Exp. Name: ***Demonstration of Working with Lists***	Date: 2025-07-22

Aim:

Write a program to demonstrate working with lists in python.

Sample Test Case:

Input:

Enter a list of elements (separated by commas): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Output:

List of elements: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Length of list: 10
Minimum value: 1
Maximum value: 10
Sum of values: 55
Sorted list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Source Code:

list.py

```
data=input("Enter a list of elements (separated by commas): ")
num_list= [int(x) for x in data.split(",")]

print(f"List of elements: {num_list}")
print(f"Length of list: {len(num_list)}")
print(f"Minimum value: {min(num_list)}")
print(f"Maximum value: {max(num_list)}")
print(f"Sum of values: {sum(num_list)}")
print(f"Sorted list: {sorted(num_list)}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a list of elements (separated by commas):
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
List of elements: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Length of list: 10
Minimum value: 1

Maximum value: 10
Sum of values: 55
Sorted list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Test Case - 2
User Output
Enter a list of elements (separated by commas):
3, 8, 2, 9, 5
List of elements: [3, 8, 2, 9, 5]
Length of list: 5
Minimum value: 2
Maximum value: 9
Sum of values: 27
Sorted list: [2, 3, 5, 8, 9]

S.No: 28	Exp. Name: ***Tuple Creation***	Date: 2025-08-12

Aim:

Write a Python program that prompts the user to enter details for two members and creates tuples for each member consisting of (name (string), age (int), address (string), college (string)). Concatenate these tuples into a single tuple and print the concatenated tuple.

Input Format:

- The first line should take user input with Name, Age, Address, and College as space-separated values for Member 1.
- The second line should take user input with Name, Age, Address, and College as space-separated values for Member 2.

Output Format:

- In the first line, print the tuple for Member 1.
- Print the tuple for Member 2 in the next line.
- Finally, print the concatenated tuple of both members.

Sample Test Case:

Input:

```
John 12 Tampa St.Joseph
Jane 15 Florida St.Mary
```

Output:

```
('John', 12, 'Tampa', 'St.Joseph') // Member 1
('Jane', 15, 'Florida', 'St.Mary') // Member 2
('John', 12, 'Tampa', 'St.Joseph', 'Jane', '15', 'Florida', 'St.Mary') //
Concatenated Tuple
```

Source Code:

stdTup.py

```
member1_input = input().split()
member1 = (member1_input[0], int(member1_input[1]),
member1_input[2],member1_input[3])
member2_input = input().split()
member2 =
(member2_input[0],int(member2_input[1]),member2_input[2],member2_input[3])
print(member1)
print(member2)
concatenated = member1 + member2
print(concatenated)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
John 12 Tampa St.Joseph
Jane 15 Florida St.Mary
`('John', 12, 'Tampa', 'St.Joseph')`
`('Jane', 15, 'Florida', 'St.Mary')`
`('John', 12, 'Tampa', 'St.Joseph', 'Jane', 15, 'Florida', 'St.Mary')`

Test Case - 2
User Output
Ram 22 Delhi IITDelhi
Varun 24 Mumbai NITWarangal
`('Ram', 22, 'Delhi', 'IITDelhi')`
`('Varun', 24, 'Mumbai', 'NITWarangal')`
`('Ram', 22, 'Delhi', 'IITDelhi', 'Varun', 24, 'Mumbai', 'NITWarangal')`

S.No: 30	Exp. Name: ***Vowel Counter***	Date: 2025-08-12

Aim:

Write a Python program to count the number of vowels in a given string without using control flow statements.

Input Format:

The program should prompt the user to enter a string.

Output Format:

Display the number of vowels present in the entered string.

Source Code:

vowel.py

```
def count_vowels(s):
    return sum(map(s.lower().count, "aeiou"))
user_input = str(input())
print(count_vowels(user_input))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
BCDFGHJKL
0

Test Case - 2
User Output
CodeTantra
4

Test Case - 3
User Output
Python Programming

S.No: 31	Exp. Name: ***Key Lookup in Dictionary***	Date: 2025-08-19

Aim:

Write a Python program that checks if a given key exists in a dictionary.

Input Format:

- The first line of input contains space separated strings that represent the keys of a dictionary.
- The second line contains space separated strings that represent the values of a dictionary.
- If the number of keys and values are not equal, the program should immediately print "Invalid" and terminate without taking the third input.
- Otherwise, the third line contains the key (as a string) to check whether it is present in the dictionary or not.

Output Format:

- If the number of keys and values are not equal, print "Invalid".
- Otherwise, the first line of the output should print the dictionary.
- Next, print "Exist" if the key is present in the dictionary, or "Does not Exist" if the key is not present in the dictionary.

Note: Refer to the visible test cases to strictly match the input/output format.

Source Code:

keyExist.py

```
keys=input().split()
values=input().split()
if len(keys)!=len(values):
        print("Invalid")
else:
        check_key=input()
        my_dict = dict(zip(keys,values))
        print(my_dict)
        if check_key in my_dict:
                print("Exist")
        else:
                print("Does not Exist")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1 2 3 4 5
A B C D E
5
{'1': 'A', '2': 'B', '3': 'C', '4': 'D', '5': 'E'}
Exist

Test Case - 2
User Output
10 20 30 40 50 60 70 80
Ten Twenty Thirty Forty Fifty Sixty
Invalid

Test Case - 3
User Output
John Reena Sara David
24 32 18 41
Jane
{'John': '24', 'Reena': '32', 'Sara': '18', 'David': '41'}
Does not Exist

S.No: 32	Exp. Name: ***Add Key Value Pair to Dictionary***	**Date: 2025-09-01**

Aim:

Write a Python program that adds a new key-value pair to an existing dictionary.

Input Format:

- The first line contains space-separated strings that represent the keys of a dictionary.
- The second line contains space-separated strings that represent the values of the dictionary.
- The third line contains two space-separated strings that represent the new key and value to be added to the existing dictionary.

Output Format:

- First line should display the original dictionary before adding the new key-value pair.
- Second line should display the updated dictionary after adding the new key-value pair.

Note:

- If the number of keys and values does not match, display "Invalid" without taking the key-value pair as input.

Source Code:

keyValue.py

```
keys = input().split()
values = input().split()

if len(keys) != len(values):
        print("Invalid")
else:
        new_input=input().split()
        new_key=new_input[0]
        new_value=new_input[1]
        dictionary = dict(zip(keys, values))
        print(dictionary)

        dictionary[new_key] = new_value

        print(dictionary)
```

Test Case - 1
User Output
Mobile Laptop Adopter Headset
10000 50000 2000
`Invalid`

Test Case - 2
User Output
thirty fifty seventy ninety forty eighty
30 50 70 90 40 80
four 4
`{'thirty': '30', 'fifty': '50', 'seventy': '70', 'ninety': '90', 'forty': '40', 'eighty': '80'}`
`{'thirty': '30', 'fifty': '50', 'seventy': '70', 'ninety': '90', 'forty': '40', 'eighty': '80', 'four': '4'}`

S.No: 33	Exp. Name: ***Sum of all Items in a Dictionary***	**Date: 2025-08-12**

Aim:

Write a program that allows the user to create a dictionary by entering key-value pairs. Calculate and print the sum of all values in the dictionary.

Input Format:

- First, the user is prompted to enter an integer size, which specifies the number of key-value pairs to be added to the dictionary.
- For each key-value pair:

1. The first line contains the key (string).
2. The second line contains the value (integer).

Output Format:

- Print the sum of all values in the dictionary.

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

Source Code:

sumOfValues.py

```
size = int(input("size:"))
my_dict = {}
total = 0
for i in range(size):
        key = input("key:")
        value = int(input("value:"))
        my_dict[key] = value
        total += value
print(f"Sum:{total}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
size:
1
key:
ninety

90
Sum:90

Test Case - 2
User Output
size:
4
key:
five
value:
5
key:
six
value:
6
key:
twelve
value:
12
key:
sixty
value:
60
Sum:83

S.No: 36	Exp. Name: ***Sort Contents of File***	Date: 2025-08-31

Aim:

Develop a program to sort the contents of a text file and write the sorted contents into a separate text file and print the output.

Use string methods `strip()`, `len()`, list methods `sort()`, `append()`, and file methods `open()`, `readlines()`, and `write()`.

Input format:

- The program will prompt the user to enter the name of an input file. This file should contain lines of text that need to be sorted.

Output format:

- The program will create a new file called **sorted_output.txt**.
- The output file will contain the lines from the input file sorted in ascending order (case-insensitive), with all words converted to lowercase, and then print the sorted contents of the file to the console.

Refer to shown test cases to match with the input and output format.

Source Code:

sortLower.py

```
# write your code here...


filename = input()

try:
    f = open(filename, "r")
    lines = f.readlines()
    f.close()

    words = []
    for line in lines:
        words.append(line.strip().lower())

    words.sort()

    out = open("sorted_output.txt", "w")
    for w in words:
        out.write(w + "\n")
    out.close()

    # print the sorted contents
    f = open("sorted_output.txt", "r")
    print(f.read(), end="")
    f.close()

except FileNotFoundError:
    print("File not found")
```

file1.txt

```
banana
apple
orange
```

file2.txt

```
aim
air
at
```

input.txt

```
Insert
text
here
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
file1.txt
apple
banana
orange

Test Case - 2
User Output
inputFile.txt
File not found

S.No: 38	Exp. Name: ***Count the Sentences, Words and Characters in a File***	**Date: 2025-09-01**

Aim:

Write a Python program that reads a file and prints the number of lines, words, and characters.

Input Format:

- The input will be a single string, which is the name of the file to be read.

Output Format:

The output will be three separate lines, each containing a count:

- The first line will contain the number of lines in the file, prefixed with "Sentences: "
- The second line will contain the number of words in the file, prefixed with "Words: "
- The third line will contain the number of characters in the file, prefixed with "Chars: "

Source Code:

Count.py

```
fp = input("file name:")

try:
        with open(fp, "r") as f:
                text = f.read()

    # Count sentences = number of lines
        sentences = text.count("\n") + (1 if text.strip() != "" else
0)

    # Count words
        words = len(text.split())

    # Count characters
        chars = len(text)
        print("Sentences:", sentences)
        print("Words:", words)
        print("Chars:", chars)

except FileNotFoundError:
        print("File not found")
```

input1.txt

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Everything matters.
```

input2.txt

```
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
Now is better than never.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
```

test1.txt

```
CodeTantra
Start coding in 60 mins
```

test3.txt

```
Count the sentences in the file.
Count the words in the file.
Count the characters in the file.
```

test2.txt

```
Hydrofoil is an underwater fin with a falt or curved wing-like
surface that is designed to lift a moving boat or ship
by means of the reaction upon its surface
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
file name:
input1.txt
Sentences: 8
Words: 34
Chars: 228

Test Case - 2
User Output
file name:
input2.txt
Sentences: 7
Words: 51
Chars: 306

S.No: 39	Exp. Name: ***Array Operations***	Date: 2025-09-01

Aim:

Write a Python program to create an array of strings entered by the user and perform the following operations sequentially:

- Display the array.
- Append an item to the array.
- Insert an item at a specified position in the array.
- Reverse the order of items in the array.

Input Format:

- The first line contains space-separated string values to initialize the array.
- The second line contains a single string to append to the array.
- The third line contains an integer representing the position at which to insert the value.
- The fourth line contains a single string value to insert into the array.

Output Format:

- Display the created array.
- Display the array after appending the value.
- Display the array after inserting the value at the specified position.
- Display the reversed array.

Note: The index position to insert the value should be less than or equal to the length of the array. Otherwise, print "Invalid index".

Refer to the shown test cases to match the input and output format with the test cases.

Source Code:

arrayOp.py

```
# Type Content here...
# Program to perform array operations

# Step 1: Take initial array input
arr = input().split()

# Step 2: Take append value
append_val = input()

# Step 3: Take insert position
pos = int(input())

# Step 4: Take insert value
insert_val = input()

# Display original array
print(arr)

# Append operation
arr.append(append_val)
print(arr)

# Insert operation
if pos <= len(arr):
    arr.insert(pos, insert_val)
    print(arr)
else:
    print("Invalid index")

# Reverse operation
arr.reverse()
print(arr)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
12 23 34 45 56 67
85
2
69

['12', '23', '34', '45', '56', '67']
['12', '23', '34', '45', '56', '67', '85']
['12', '23', '69', '34', '45', '56', '67', '85']
['85', '67', '56', '45', '34', '69', '23', '12']

Test Case - 2
User Output
a b c d e f g h
k
5
o
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'k']
['a', 'b', 'c', 'd', 'e', 'o', 'f', 'g', 'h', 'k']
['k', 'h', 'g', 'f', 'o', 'e', 'd', 'c', 'b', 'a']

Test Case - 3
User Output
hi hello how are you
fine
7
you
['hi', 'hello', 'how', 'are', 'you']
['hi', 'hello', 'how', 'are', 'you', 'fine']
Invalid index
['fine', 'you', 'are', 'how', 'hello', 'hi']

S.No: 40	Exp. Name: ***Matrix Addition***	**Date: 2025-09-01**

Aim:

Write a Python program to add two matrices. First, prompt the user to input the dimensions of two matrices ($r1$, $c1$) for the first matrix, and then allow the user to input the elements of each matrix. Then prompt the user to enter dimensions ($r2$, $c2$) for the second matrix and allow the user to enter elements. If the matrices have compatible dimensions ($r1 == r2$ and $c1 == c2$), compute their sum; otherwise, indicate that matrix addition is not possible due to incompatible dimensions.

Input Format:

- The first line contains two space-separated integers representing the number of rows ($r1$) and columns($c1$) for matrix1
- Next, $r1$ lines contain $c1$ number of integers separated by a space, representing the elements of matrix 1
- The next line contains two space-separated integers representing the number of rows ($r2$) and columns ($c2$) for matrix2
- Next, $r2$ lines contain $c2$ number of integers separated by a space, representing the elements of matrix 2

Output Format:

- If the matrices have compatible dimensions ($r1 == r2$ and $c1 == c2$), print the sum of corresponding elements of the matrices; otherwise, print "Invalid".

Source Code:

matrix.py

```
# Type Content here...
# Matrix Addition Program

# First matrix dimensions
r1, c1 = map(int, input().split())
mat1 = [list(map(int, input().split())) for _ in range(r1)]

# Second matrix dimensions
r2, c2 = map(int, input().split())
mat2 = [list(map(int, input().split())) for _ in range(r2)]

# Check if dimensions match
if r1 == r2 and c1 == c2:
        for i in range(r1):
                row = []
                for j in range(c1):
                        row.append(mat1[i][j] + mat2[i][j])
                print(*row)
else:
        print("Invalid")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
2 3
1 2 3
4 5 6
2 3
7 8 9
0 1 2
8 10 12
4 6 8

Test Case - 2
User Output
2 2
1 2

2 3
2 3
1 2 3
0 2 3
Invalid

S.No: 41	Exp. Name: ***Matrix Multiplication***	**Date: 2025-09-01**

Aim:

Write a Python function **matrix_multiplication()** that performs matrix multiplication of two matrices and prints the resulting matrix. If the matrices cannot be multiplied, the function should print an appropriate error message.

Input Format:

- The dimensions of the first matrix ($r1$ $c1$), where $r1$ is the number of rows and $c1$ is the number of columns.
- The elements of the first matrix are provided row by row.
- The dimensions of the second matrix ($r2$ $c2$), where $r2$ is the number of rows and $c2$ is the number of columns.
- The elements of the second matrix are provided row by row.

Output Format:

- Print the resulting matrix after multiplication, where each element of the matrix is separated by a space; otherwise, print "Invalid".

Note: The number of columns of the first matrix must be equal to the number of rows of the second matrix for matrix multiplication.

Source Code:

matrixmul.py

```
def matrix_multiplication():
        r1, c1 = map(int, input().split())
        mat1 = [list(map(int, input().split())) for _ in range(r1)]
        r2, c2 = map(int, input().split())
        mat2 = [list(map(int, input().split())) for _ in range(r2)]

    # Check condition for multiplication
        if c1 != r2:
                print("Invalid")
                return

    # Initialize result matrix with zeros
        result = [[0] * c2 for _ in range(r1)]

    # Matrix multiplication
        for i in range(r1):
                for j in range(c2):
                        for k in range(c1):
                                result[i][j] += mat1[i][k] * mat2[k]
[j]

    # Print result matrix
        for row in result:
                print(*row)


# Call the function to run matrix multiplication
matrix_multiplication()
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
2 3
1 2 3
4 5 6
3 4
1 2 3 4
5 6 7 8
9 0 1 2

Test Case - 2
User Output
2 2
-3 4
4 -7
2 2
-4 5
-6 7
-12 13
26 -29

Test Case - 3
User Output
2 2
6 12
-3 7
3 2
12 23 34
12 34 54
4 26 19
Invalid

S.No: 42	Exp. Name: ***Transpose of Matrix***	Date: 2025-09-01

Aim:

Write a Python program to find the transpose of a matrix.

Input Format:

- The first line contains two space-separated integers representing the number of rows (R) and columns (C) for the matrix.
- Next, R lines contain C number of integers separated by a space, representing the elements of the matrix.

Output Format:

- First, print "Transpose of the matrix:"
- Next, print the transpose of the matrix, with each row on a new line and elements space-separated.

Source Code:

transpose.py

```
# Input rows and columns
r, c = map(int, input().split())

# Input matrix
matrix = [list(map(int, input().split())) for _ in range(r)]

# Transpose
transpose = [[matrix[j][i] for j in range(r)] for i in range(c)]

# Output

for row in transpose:
        print(*row)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
2 4
10 20 30 40
12 23 45 52

10 12
20 23
30 45
40 52

Test Case - 2
User Output
3 5
-10 -20 -30 -40 -50
-11 -22 -33 -44 -55
-13 -15 -17 -18 -19
-10 -11 -13
-20 -22 -15
-30 -33 -17
-40 -44 -18
-50 -55 -19

S.No: 43	Exp. Name: ***Geometry Class***	Date: 2025-09-01

Aim:

Write a Python program that defines a base class **Shape** with two methods, **area** and **perimeter**, that are meant to be overridden by subclasses. Define three subclasses: **Circle**, **Triangle**, and **Square**, each of which implements the **area** and **perimeter** methods. The program should then create instances of these subclasses based on user input and print their areas and perimeters, formatted to two decimal places.

Input Format:

- The first line contains a single float representing the radius of a circle.
- The second line contains three space-separated floats representing the sides of a triangle.
- The third line contains a single float representing the side length of a square.

Output Format:

- Print the area and perimeter of the circle on one line, separated by a space, formatted to two decimal places.
- Print the area and perimeter of the triangle on the next line, separated by a space, formatted to two decimal places.
- Print the area and perimeter of the square on the last line, separated by a space, formatted to two decimal places.

Note:

- Triangle sides must satisfy the triangle inequality theorem: the sum of any two sides must be greater than the third side, ensuring the formation of a valid triangle.
- Area of traingle $ABC = \sqrt{[s \times (s-a) \times (s-b) \times (s-c)]}$
- $s = \frac{(side1 + side2 + side3)}{2}$

Source Code:

geometry1.py

```
import math

# Base class
class Shape:
        def area(self):
                raise NotImplementedError("Subclass must implement
area method")
        def perimeter(self):
                raise NotImplementedError("Subclass must implement
perimeter method")

#Circle sub class
class Circle(Shape):
        def __init__(self, radius):
                self.radius = radius
        def area(self):
                return math.pi * self.radius ** 2
        def perimeter(self):
                return 2 * math.pi * self.radius


# Triangle subclass
class Triangle(Shape):
        def __init__(self, a, b, c):
                self.a = a
                self.b = b
                self.c = c
        def area(self):
                s = (self.a + self.b + self.c) / 2
                return math.sqrt(s * (s - self.a) * (s - self.b) * (s
- self.c))
        def perimeter(self):
                return self.a + self.b + self.c


# Square subclass
class Square( Shape):
        def __init__(self, side):
                self.side = side
        def area(self):
                return self.side ** 2
        def perimeter(self):
                return 4 * self.side
if __name__ == "__main__":
        radius = float(input().strip())
        sides = list(map(float, input().split()))
```

```
side_square = float(input().strip())
circle = Circle(radius)
triangle = Triangle(*sides)
square = Square(side_square)
print(f"{circle.area():.2f} {circle.perimeter():.2f}")
print(f"{triangle.area():.2f} {triangle.perimeter():.2f}")
print(f"{square.area():.2f} {square.perimeter():.2f}")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
12
4 5 6
4
452.39 75.40
9.92 15.00
16.00 16.00

Test Case - 2
User Output
0
0 0 0
0
0.00 0.00
0.00 0.00
0.00 0.00

Test Case - 3
User Output
5.0
3.0 4.0 5.0
7.5
78.54 31.42
6.00 12.00

```
56.25 30.00
```

S.No: 46	Exp. Name: ***Slicing, Boolean Indexing of Array***	**Date: 2025-09-02**

Aim:

Write a python program to demonstrate basic slicing, integer and Boolean indexing of array using NumPy operations. Your program should handle basic slicing, integer indexing, and boolean indexing operations on the array.

Input & Output Format:

- First line: Two integers specifying number of rows (rows) and columns (cols) of the array (space separated).
- Then, display the NumPy array after constructing it from the provided input elements row wise space separated.
- Next line: Two integers (start and end) specifying the range of rows to slice from the array (space separated).
- Then, display the sliced portion of the array based on the start and end indices. If the indices are invalid, output "Invalid". (The start index should be non-negative and less than the number of rows, and end index (end) should be within the range of rows to consider it as invalid).
- Next line: An integer (index) specifying the position to access an element using integer indexing.
- Display the element in the array at the specified index. If the index is invalid, output "Invalid"(index should be a non-negative integer less than the number of rows in the array).
- Next line: A string of space-separated integers (0 or 1) representing boolean indices for selecting rows from the array.
- Display the boolean index array used for selection followed by elements from the array that correspond to True values in the boolean index array.

Source Code:

array_operations.py

```
import numpy as np
# Read rows and columns
rows, cols = map(int, input().split())
# Read elements row-wise
elements = []
for _ in range(rows):
        row = list(map(int, input().split()))
        if len(row) !=cols:
                print("Invalid")
                exit()
        elements.extend(row)
# Construct NumPy array
arr = np.array(elements).reshape(rows, cols)
print(arr)

# Slicing
start, end = map(int, input().split())
if 0 <= start < rows and 0 < end <= rows and start < end:
        print(arr[start:end])
else:
        print("Invalid")

# Integer Indexing
index = int(input())
if 0 <= index < rows:
        print(arr[index])
else:
        print("Invalid")
# Boolean Indexing
bool_indices = list(map(int, input().split()))
if len(bool_indices)==rows and all(b in (0,1) for b in bool_indices):
        bool_array=np.array(bool_indices,dtype=bool)
        print(bool_array)
        print(arr[bool_array])
else:
        print("Invalid")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3 4

1 2 3 4
5 6 7 8
9 10 11 12
[[1 2 3 4]
[5 6 7 8]
[9 10 11 12]]
0 2
[[1 2 3 4]
[5 6 7 8]]
1
[5 6 7 8]
1 0 1
[True False True]
[[1 2 3 4]
[9 10 11 12]]

Test Case - 2
User Output
0 0
[]
0 0
Invalid
0
Invalid
0
Invalid

S.No: 47	Exp. Name: ***Scatter Plot using Pandas Dataframe***	**Date: 2025-10-19**

Aim:

Create a scatter plot to visualize the relationship between two columns from a dataset represented as a dictionary. Convert the dictionary into a Pandas DataFrame for this purpose.

Input Format:

- Prompt the user to input two column names (strings) separated by space.

Output Format:

- Display a scatter plot showing the relationship between the selected columns.
- Ensure the plot includes appropriate labels, title, legend, grid, and customization (such as color and marker style).

Plot Settings:

- Use blue color to set the color of the markers
- Marker Style: Use marker='o' to set the marker style
- Use "Scatter" as label for the scatter plot in the legend.
- Set "Scatter Plot" as the title of the plot.
- Use column1 and column2 as the labels for the x-axis and y-axis, respectively.
- Display the legend.
- Display the grid on the plot for better visualization.

Note: Refer to the data set that has been provided to you in the editor.

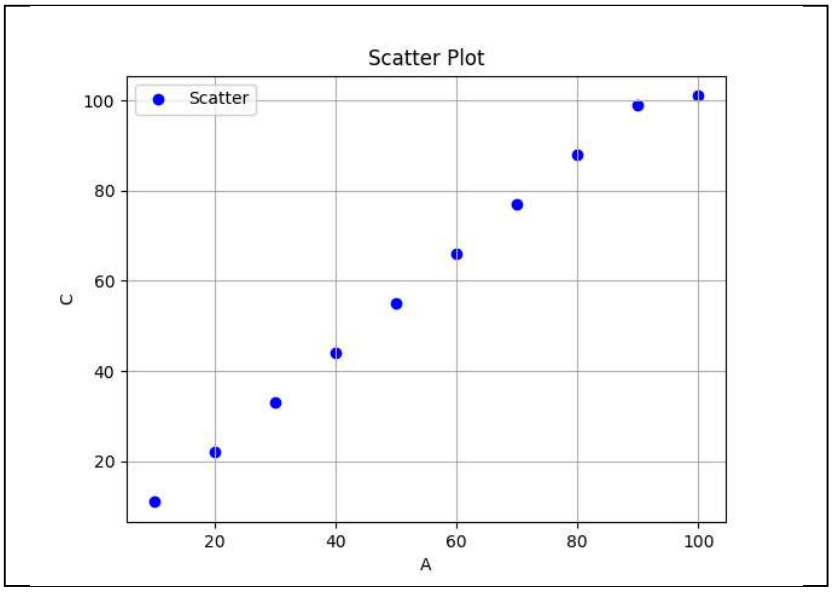
Source Code:

scatter.py

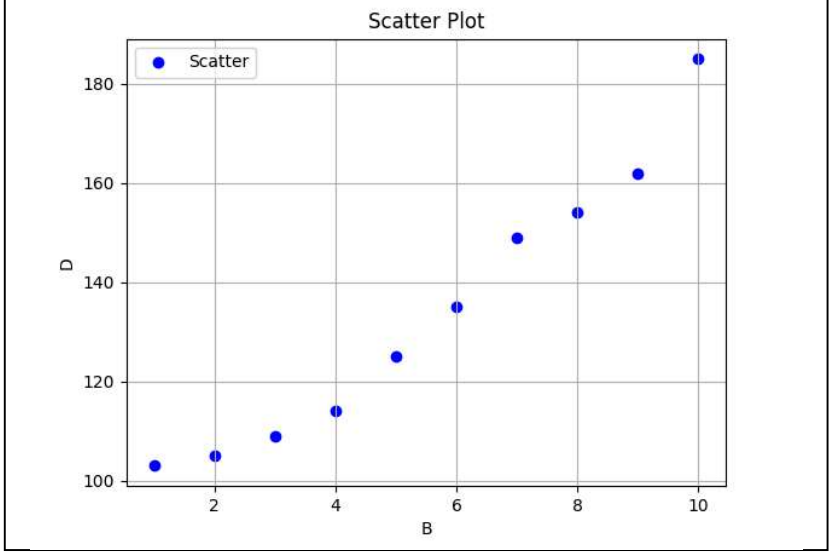
```
import pandas as pd
import matplotlib.pyplot as plt

# Given DataFrame
std_marks = {
    'A': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    'B': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'C': [11, 22, 33, 44, 55, 66, 77, 88, 99, 101],
    'D': [103, 105, 109, 114, 125, 135, 149, 154, 162, 185],
    'E': [52, 86, 75, 41, 32, 71, 68, 94, 73, 69]
}

# Construct a Pandas DataFrame using the provided item data


# Filter the DataFrame to include only relevant columns for
visualization

df=pd.DataFrame(std_marks)
col1,col2=input().split()
plt.scatter(df[col1],df[col2],color='blue',marker='o',label="Scatter")
plt.title("Scatter plot")
plt.xlabel(col1)
plt.ylabel(col2)
plt.legend()
plt.grid(True)
plt.show()
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
A C



Test Case - 2
User Output
B D



S.No: 48	Exp. Name: ***Numpy Array***	Date: 2025-10-19

Aim:

Write a Python program that prompts the user to enter numeric elements for a NumPy array. The program should create the NumPy array using the entered elements and display it.

Input Format:

- A single line of space-separated numeric values entered by the user.

Output Format:

- The NumPy array created from the user's input.

Note:

- Refer to the sample test cases for better understanding regarding input and output format.

Source Code:

arrayNumpy.py

```
import numpy as np
values=list(map(int,input().split()))
arr=np.array(values)
print(arr)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1 2 3 4 5
[1 2 3 4 5]

Test Case - 2
User Output
10 20 30 40 50 60 70 80 90
[10 20 30 40 50 60 70 80 90]

Test Case - 3
User Output
-1 -2 -3 -4 5 6 7 8
[-1 -2 -3 -4 5 6 7 8]

S.No: 49	Exp. Name: ***Numpy Array Operations***	Date: 2025-10-19

Aim:

Write a Python program to create a NumPy array based on user input and display the following:

- The created NumPy array.
- The number of dimensions of the array using `.ndim`.
- The shape of the array using `.shape`.
- The total number of elements in the array using `.size`.

Assume all input elements are valid integers.

Input Format:

- The first line contains two space-separated integers, r and c, representing the number of rows and the number of columns.
- The next r lines contain c space-separated integers representing the elements of the array, entered row by row.

Output Format:

- The created NumPy array (printed as a 2D array).
- An integer representing the number of dimensions of the array.
- A tuple representing the shape of the array.
- An integer representing the total number of elements in the array.

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

Source Code:

numpyarr.py

```
import numpy as np
r,c=map(int,input().split())
if r==0 or c==0:
        arr=np.array([]).reshape(r,c)
else:
        data=[list(map(int,input().split())) for _ in range(r)]
        arr=np.array(data)
print(arr)
print(arr.ndim)
print(arr.shape)
print(arr.size)
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3 4
1 2 3 4
5 6 7 8
9 10 11 12
[[1 2 3 4]
[5 6 7 8]
[9 10 11 12]]
2
(3, 4)
12

Test Case - 2
User Output
0 0
[]
2
(0, 0)
0

Test Case - 3
User Output
2 3
10 20 30
40 50 60
[[10 20 30]
[40 50 60]]
2
(2, 3)
6

S.No: 50	Exp. Name: ***Numpy Array Statistics***	Date: 2025-10-19

Aim:

Write a Python program that prompts the user to enter elements for a NumPy array. The program should compute and print the minimum value, maximum value and sum of elements of the array using NumPy functions.

Input Format:

- The program expects the user to enter elements of the array separated by spaces.

Output Format:

- First, print the original NumPy array as integers.
- Next, print the minimum value found in the array as an integer.
- Then, display the maximum value present in the array as an integer.
- Finally, show the sum of all elements within the array as an integer.

Source Code:

arrayStat.py

```
import numpy as np
values=list(map(int,input().split()))
arr=np.array(values)
print(arr)
print(np.min(arr))
print(np.max(arr))
print(np.sum(arr))
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1 2 3 4 5 6 7 8 9
[1 2 3 4 5 6 7 8 9]
1
9
45

User Output
-10 -20 -30 -40 -50
[-10 -20 -30 -40 -50]
-50
-10
-150

S.No: 51	Exp. Name: ***Numpy, pandas operations using dictionary***	**Date: 2025-10-19**

Aim:

Convert the given dataset into a Pandas DataFrame representing students (A, B, C, D, E) and their marks in ten different subjects. Each student's data is structured as follows:

- Keys represent student names (A, B, C, D, E).
- Values are lists containing the marks for ten subjects respectively.

Define a function that allows the user to perform a custom data selection operation on this DataFrame. The function should:

- Take user input for the student's name (column) and a minimum marks threshold (condition_value).
- Apply the head() function to display the initial rows of the DataFrame.
- Filter and display the rows (students) whose marks in the specified subject (column) are greater than the provided threshold (condition_value).

Input Format:

- Prompt the user to input the student's name (A, B, C, D, E).
- Prompt the user to input the minimum marks they want to filter by in the selected subject.

Output Format:

- Display the output of the head() function applied to the Pandas DataFrame.
- Print the marks of the specified student that meet the specified criteria.

Source Code:

stdMarksDf.py

```
import pandas as pd

# Predefined dictionary with at least five keys, each representing a
student's subject marks
std_marks = {
    'A': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    'B': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'C': [11, 22, 33, 44, 55, 66, 77, 88, 99, 101],
    'D': [103, 105, 109, 114, 125, 135, 149, 154, 162, 185],
    'E': [52, 86, 75, 41, 32, 71, 68, 94, 73, 69]
}
df=pd.DataFrame(std_marks)
student_name=input()
condition_value=float(input())
print(df.head())
if student_name in df.columns:
        selected_data=df[df[student_name]>condition_value]
[student_name]
if not selected_data.empty:
    print(selected_data)
else:
    print("Invalid")
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
A
35
A B C D E
0 10 1 11 103 52
1 20 2 22 105 86
2 30 3 33 109 75
3 40 4 44 114 41
4 50 5 55 125 32
3 40
4 50
5 60
6 70
7 80

```
Name: A, dtype: int64
```

Test Case - 2

User Output

```
E
100
    A  B   C    D   E
0  10  1  11  103  52
1  20  2  22  105  86
2  30  3  33  109  75
3  40  4  44  114  41
4  50  5  55  125  32
Invalid
```

Test Case - 3

User Output

```
C
62.83
    A  B   C    D   E
0  10  1  11  103  52
1  20  2  22  105  86
2  30  3  33  109  75
3  40  4  44  114  41
4  50  5  55  125  32
5     66
6     77
7     88
8     99
9    101
Name: C, dtype: int64
```