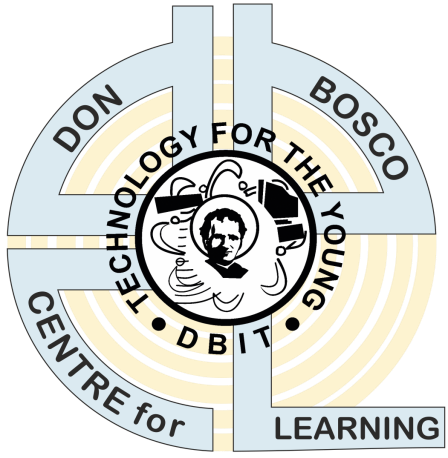
# Python Lab Journal



Name: Sahil Vikas Khude

Roll No. 31

Course/Lab: \_\_\_\_\_\_\_\_\_\_\_\_ (CODE \_\_\_\_\_\_\_)

Semester: IV

Academic Year: 2024-25

## Subject In-charge: Prof. Shiv Negi

Department of Information Technology

The Bombay Salesian Society’s

Don Bosco Institute of Technology, Mumbai 400070.

Index Page

**Title: Print the grades of students after accepting the marks for 5 subjects from the user**

**Theory:**

In this program, we are taking input of marks for 5 subjects from the user using a for loop and storing them in a list. We then calculate the total and average of these marks using the built-in sum() function and basic arithmetic.

Based on the average marks, we assign grades using if-elif-else conditional statements. These conditions help us decide what grade the student falls into. Finally, we print the total, average, and the grade.

Key Python concepts used here:

* Input and typecasting
* List to store values
* Loops (for loop)
* Conditional statements (if-elif-else)
* Basic arithmetic operations

**Source Code:**

marks = []

for i in range(1, 6):

    mark = float(input(f'Enter marks of subject { i }: '))

    marks.append(mark)

total = 0

for i in marks:

    total = total + i

average = total / 5

if average >= 90:

    grade = 'A+'

elif average >= 80:

    grade = 'A'

elif average >= 70:

    grade = 'B+'

elif average >= 60:

    grade = 'B'

elif average >= 50:

    grade = 'C'

elif average >= 40:

    grade = 'D'

else:

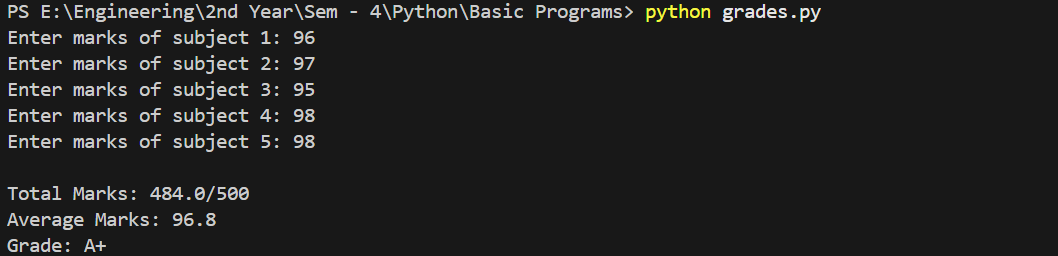
    grade = 'F'

print(f"\nTotal Marks: {total}/500")

print(f"Average Marks: {average}")

print(f"Grade: {grade}")

**Sample Output:**

****

**Title: Accept the range from the user and then display all prime numbers between the given range.**

**Theory:**

This program finds all prime numbers between a user-defined range. A prime number is a number greater than 1 that has no divisors other than 1 and itself.

The user inputs the starting and ending values of the range. The program then uses a for loop to check each number in this range. For each number, we use another loop to check if it's divisible by any number between 2 and its square root (num \*\* 0.5)—which is a shortcut to reduce unnecessary checks and improve performance.

If no divisors are found, that number is printed as a prime number. The logic is built using:

* Loops (for loop)
* Conditional statements (if-else)
* Modulo operator (%) to check divisibility
* Mathematical optimization using square root method to improve efficiency

**Source Code:**

start = int(input("Enter the starting number of the range: "))

end  = int(input("Enter the ending number of the range: "))

print(f'\n Prime numbers between {start} and {end} are: ')

for num in range(start, end+1):

    if num > 1:

        for i in range(2, int(num \*\* 0.5) + 1):

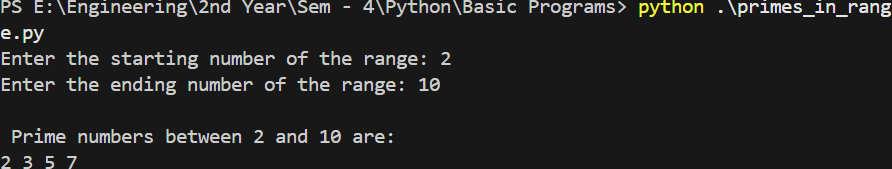
            if num % i == 0:

                break

        else:

                print(num, end=' ')

**Sample Output:**

****

**Title: Print an n digit number in reverse order.**

**Theory:**

In this program, we accept an n-digit number from the user. Instead of converting it to an integer, we take it as a string so that we can directly reverse it using string slicing. In Python, the slicing syntax [::-1] allows us to reverse any string in a simple and clean way.

This program demonstrates:

* Input handling
* String slicing technique for reversal
* Avoiding complex logic with loops by using Python’s built-in features

**Source Code:**

number = int(input("Enter a number : "))

num1 = number

print("The reverse of number", num1, "is ", end='')

while num1 != 0:

    i = num1 % 10

    print(i, end='')

    num1 //= 10

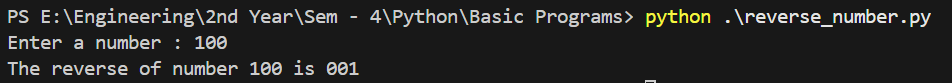
while(num1 != 0):

    i = num1 % 10

    print(i, end='')

    num1 //= 10

**Sample Output:**



**Title: Display   
 i) a pattern formed with numbers  
 ii) a pattern formed with ‘\*’  
 using nested looping.**

**Theory:**

In this program, we create two different patterns using nested loops:

1. Number Pattern:
   * For each row, we print numbers starting from 1 up to the row number.
   * This uses two for loops: the outer loop for rows and the inner loop for columns.
2. Star Pattern (\*):
   * For each row, we print that many stars.
   * Again, we use nested loops to control the structure of the triangle.

**Source Code:**

**rows = int(input("Enter the number of rows for the pattern: "))**

**print('\nNumber Pattern')**

**for i in range(1, rows + 1):**

**for j in range(1, i+1):**

**print(j, end=' ')**

**print()**

**print('\nStar Pattern: ')**

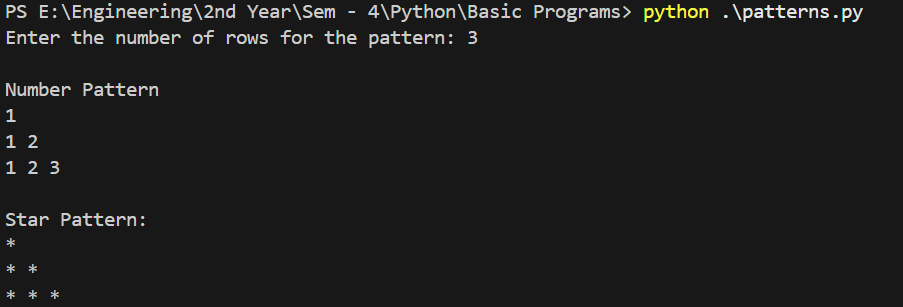
**for i in range(1, rows + 1):**

**for j in range(i):**

**print('\*', end=' ')**

**print()**

**Sample Output:**

****

**Title: Display the count of elements of different data types present in a LIST.**

**Theory:**

In this program, we use a list containing elements of multiple data types — like integers, strings, floats, booleans, lists, dictionaries, etc.

To solve this:

* We use a for loop to go through each element.
* The type() function (with \_\_name\_\_) helps identify the data type of each item.
* Lists
* Dictionaries
* Loops
* Built-in type() function
* Conditional logic

**Source Code:**

**mixed\_list = [1, 'string', 2.2, True, None, 42, 'world', False, [1,3], {'a': 1}]**

**type\_count = {}**

**for item in mixed\_list:**

**item\_type = type(item).\_\_name\_\_**

**if item\_type in type\_count:**

**type\_count[item\_type] += 1**

**else:**

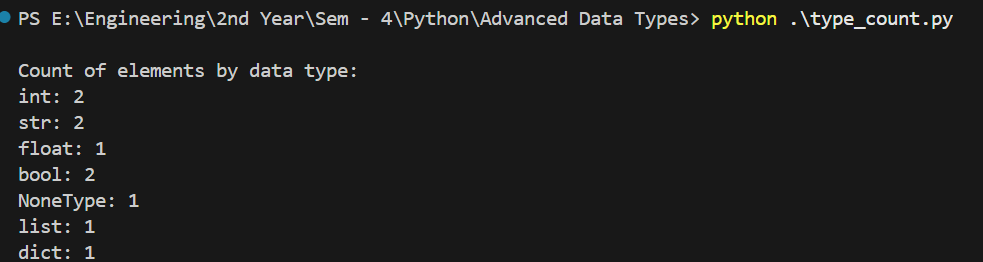
**type\_count[item\_type] = 1**

**print('\nCount of elements by data type: ')**

**for dtype, count in type\_count.items():**

**print(f'{dtype}: {count}')**

**Sample Output:**

****

**Title: Check for a given value in the LIST; display total count of occurrences along with the index positions of each occurrence.**

**Theory:**

**This program searches for a user-specified value in a list and finds:**

* **The total number of times the value appears**
* **The index positions at which it occurs**

**We use:**

* **enumerate() function to get both index and value during the loop**
* **A counter variable to track the number of occurrences**
* **A list to store the positions where matches are found**
* **An if-else condition to check and respond accordingly**

**Source Code:**

**my\_list = [10,20,30,10,40,10,50,20,10]**

**search\_value = int(input("Enter the value to search in the list: "))**

**count = 0**

**positions = []**

**for index, value in enumerate(my\_list):**

**if value == search\_value:**

**count += 1**

**positions.append(index)**

**print(f"\nThe value {search\_value} occurred {count} times.")**

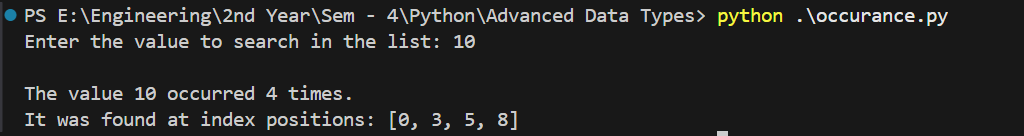
**if count > 0:**

**print("It was found at index positions:", positions)**

**else:**

**print("The value was not found in the list.")**

**Sample Output:**

****

**Title: Perform sorting of LIST elements; Press 1 for ascending order and, Press 2 for descending order.**

**Theory:**

In this program, we sort list elements without using built-in sorting methods.  
We use Bubble Sort — a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they're in the wrong order.

We implemented:

* Ascending order: by swapping if the left element is greater than the right.
* Descending order: by swapping if the left element is smaller than the right.

Key Python concepts used:

* Nested loops
* Swapping using tuple unpacking
* User input-based conditional logic

**Source Code:**

my\_list = [45, 12, 78, 34, 89, 23, 67]

print("Original List: ", my\_list)

choice = int(input("Press 1 for Ascending Order or Press 2 for Descending Order: "))

n = len(my\_list)

if choice == 1:

    for i in range(n):

        for j in range(0, n-i-1):

            if my\_list[j] > my\_list[j+1]:

                my\_list[j], my\_list[j+1] = my\_list[j+1], my\_list[j]

    print("List in Ascending Order:", my\_list)

elif choice == 2:

    for i in range(n):

        for j in range(0, n-i-1):

            if my\_list[j] < my\_list[j+1]:

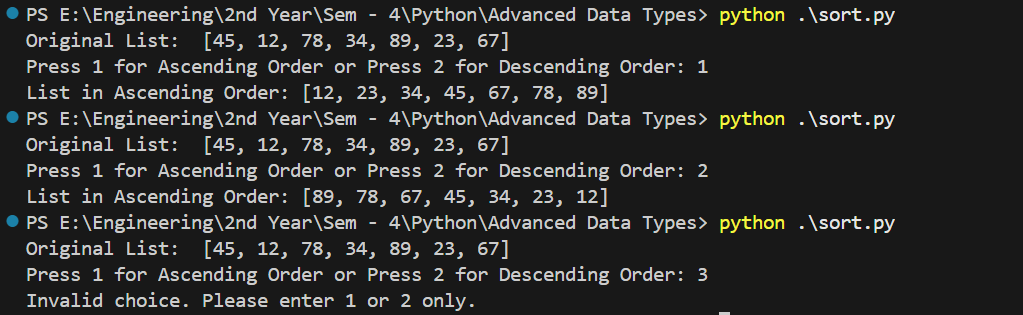
                my\_list[j], my\_list[j+1] = my\_list[j+1], my\_list[j]

    print("List in Ascending Order:", my\_list)

else:

    print("Invalid choice. Please enter 1 or 2 only.")

**Sample Output:**

****

**Title: Add elements of List2 in List1, then display the updated List.**

**Theory:**

In this program, we are merging two lists: List2 is added element-by-element into List1.

To do this:

* We use a for loop to iterate through List2
* For each element, we use the append() method of Python lists to add it to List1

This avoids using + or extend() — and manually demonstrates the process of list merging.

Key Python concepts:

* Lists
* Loops
* append() method to add individual elements

**Source Code:**

list1 = [10, 20, 30]

list2 = [40, 50, 60]

print("List1 before adding:", list1)

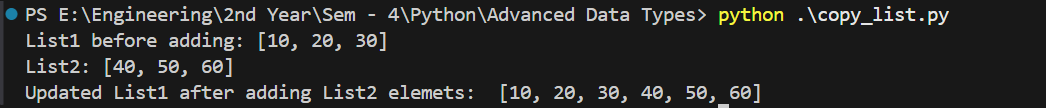
print("List2:", list2)

for element in list2:

list1.append(element)

print("Updated List1 after adding List2 elements:", list1)

**Sample Output:**

****

**Title: Demonstrate LIST comprehensions using two examples.**

**Theory:**

List Comprehension is a short and elegant way to create lists in Python.  
Instead of writing multiple lines using loops, we can write the logic in a single line.

In Example 1, we generate a list of squares using:

Python

In Example 2, we use a condition inside the list comprehension to filter even numbers:

Key Python concepts used:

* List Comprehension syntax
* Looping and conditional filtering
* Mathematical operations within list creation

**Source Code:**

end = int(input("Enter number to find squares till that number: "))

squares = [x\*\*2 for x in range(1, end + 1)]

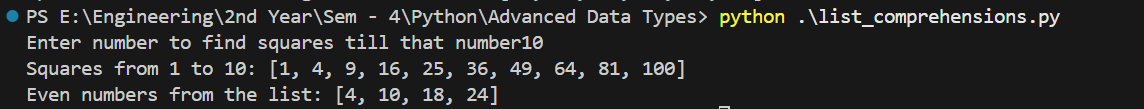
print("Squares from 1 to 10:", squares)

numbers = [1, 4, 7, 10, 15, 18, 21, 24]

even\_numbers = [num for num in numbers if num % 2 == 0]

print("Even numbers from the list:", even\_numbers)

**Sample Output:**

****

**Title: Display count of vowels, consonants, blank spaces, special symbols, and digits in a given STRING.**

**Theory:**

This program takes a string input from the user and counts different types of characters:

* Vowels (a, e, i, o, u)
* Consonants (other alphabets)
* Digits (0–9)
* Blank spaces (isspace())
* Special characters (punctuations, symbols etc.)

We use:

* String methods like .isalpha(), .isdigit(), .isspace() to classify characters
* Looping through each character for detailed analysis
* Counter variables for tracking each type

**Source Code:**

text = input("Enter a string: ")

vowels = consonants = spaces = digits = special\_chars = 0

vowel\_set = 'aeiouAEIOU'

for char in text:

    if char in vowel\_set:

        vowels += 1

    elif char.isalpha():

        consonants += 1

    elif char.isdigit():

        digits += 1

    elif char.isspace():

        spaces += 1

    else:

        special\_chars += 1

print("\nAnalysis of the given string:")

print("Vowels:", vowels)

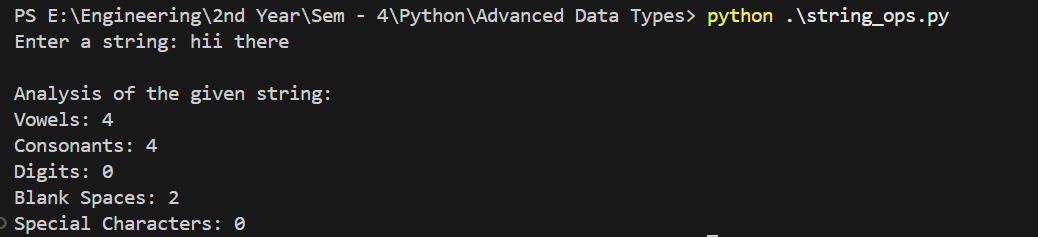
print("Consonants:", consonants)

print("Digits:", digits)

print("Blank Spaces:", spaces**)**

print("Special Characters:", special\_chars)

**Sample Output:**

****

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**

**Title:**

**Theory:**

**Source Code:**

**Sample Output:**