

PYTHON PROGRAMMING LAB



Prepared by:

Name of Student: Sakshi Kore

Roll No: 33

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Exp. No	List of Experiment
1	<p>1. Write a program to compute Simple Interest.</p> <p>2. Write a program to perform arithmetic, Relational operators.</p> <p>3. Write a program to find whether a given no is even & odd.</p> <p>4. Write a program to print first n natural number & their sum.</p> <p>5. Write a program to determine whether the character entered is a Vowel or not</p> <p>6. Write a program to find whether given number is an Armstrong Number.</p> <p>7. Write a program using for loop to calculate factorial of a No.</p>
	1.8 Write a program to print the following pattern
	i) <pre>* * * * * * * * * * * * * * *</pre>
	ii) <pre>1 2 2 3 3 3 4 4 4 4 5 5 5 5 5</pre>

	<p>iii)</p> <pre> * * * * * * * * * * * * </pre>
2	<p>2.1 Write a program that defines the list of countries that are in BRICS.</p>
	<p>2.2 Write a program to traverse a list in reverse order.</p> <ol style="list-style-type: none"> 1.By using Reverse method. 2.By using slicing
	<p>2.3 Write a program that scans the email address and forms a tuple of username and domain.</p>
	<p>2.4 Write a program to create a list of tuples from given list having number and add its cube in tuple. i/p: c= [2,3,4,5,6,7,8,9]</p>
	<p>2.5 Write a program to compare two dictionaries in Python? (By using == operator)</p>
	<p>2.6 Write a program that creates dictionary of cube of odd numbers in the range.</p>

	<p>2.7 Write a program for various list slicing operation.</p> <p>a= [10,20,30,40,50,60,70,80,90,100]</p> <ul style="list-style-type: none"> i. Print Complete list ii. Print 4th element of list iii. Print list from 0th to 4th index. iv. Print list -7th to 3rd element v. Appending an element to list. vi. Sorting the element of list. vii. Popping an element. viii. Removing Specified element. ix. Entering an element at specified index. x. Counting the occurrence of a specified element. xi. Extending list. xii. Reversing the list.
3	<p>3.1 Write a program to extend a list in python by using given approach.</p> <ul style="list-style-type: none"> i. By using + operator. ii. By using Append () iii. By using extend ()
	<p>3.2 Write a program to add two matrices.</p>
	<p>3.3 Write a Python function that takes a list and returns a new list with distinct elements from the first list.</p>
	<p>3.4 Write a program to Check whether a number is perfect or not.</p>
	<p>3.5 Write a Python function that accepts a string and counts the number of upper- and lower-case letters.</p> <pre>string_test= 'Today is My Best Day'</pre>
4	<p>4.1 Write a program to Create Employee Class & add methods to get employee details & print.</p>

	4.2 Write a program to take input as name, email & age from user using combination of keywords argument and positional arguments (*args and **kwargs) using function,
	4.3 Write a program to admit the students in the different Departments(pgdm/btech)and count the students. (Class, Object and Constructor).
	4.4 Write a program that has a class store which keeps the record of code and price of product display the menu of all product and prompt to enter the quantity of each item required and finally generate the bill and display the total amount.
	4.5 Write a program to take input from user for addition of two numbers using (single inheritance).
	4.6 Write a program to create two base classes LU and ITM and one derived class. (Multiple inheritance).
	4.7 Write a program to implement Multilevel inheritance, Grandfather→Father→Child to show property inheritance from grandfather to child.
5	4.8 Write a program Design the Library catalogue system using inheritance take base class (library item) and derived class (Book, DVD & Journal) Each derived class should have unique attribute and methods and system should support Check in and check out the system. (Using Inheritance and Method overriding)
5	5.1 Write a program to create my_module for addition of two numbers and import it in main script.
	5.2 Write a program to create the Bank Module to perform the operations such as Check the Balance, withdraw and deposit the money in bank account and import the module in main file.
	5.3 Write a program to create a package with name cars and add different modules (such as BMW, AUDI, NISSAN) having classes and functionality and import them in main file cars.

6	6.1 Write a program to implement Multithreading. Printing “Hello” with one thread & printing “Hi” with another thread.
7.	7.1 Write a program to use ‘whether API’ and print temperature of any city, also print the sunrise and sunset times for the same humidity of that area.
	7.2 Write a program to use the ‘API’ of crypto currency.

Experiment No: 1.1

Title: Write a program to compute Simple Interest.

Theory:

Calculates the simple interest (SI) based on the given principle amount, rate of interest, and time period. Formula for SI = $P \times R \times T / 100$

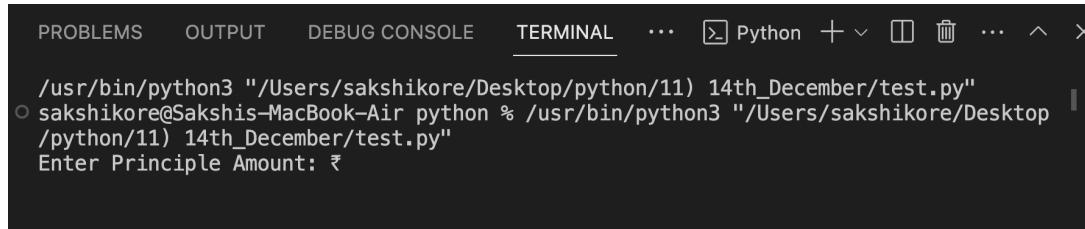
Code:

```
principle = int(input("Enter Principle Amount: ₹"))
rate = int(input("Enter Rate Of Interest: "))
time = int(input("Enter Time Period: "))

si = (principle * rate * time) / 100

print("The Simple Interest is:", si)
```

Output: (screenshot)



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ... Python + ∨ × ... ^ ×
/usr/bin/python3 "/Users/sakshikore/Desktop/python/11) 14th_December/test.py"
○ sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/
/python/11) 14th_December/test.py"
Enter Principle Amount: ₹1000
Simple Interest is: 200.0
```

Test Case: Any two (screenshot)

- sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/
/python/11) 14th_December/test.py"
Enter Principle Amount: ₹800
Enter Rate Of Interest: 5
Enter Time Period: 4
The Simple Interest is: 160.0

- sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/
/python/11) 14th_December/test.py"
Enter Principle Amount: ₹1350
Enter Rate Of Interest: 6
Enter Time Period: 3
The Simple Interest is: 243.0

Conclusion:

Simple interest is calculated based on the values entered by the user with respect to the formula mentioned in the code.

Experiment No: 1.2

Title: Write a program to perform arithmetic, Relational operators.

Theory:

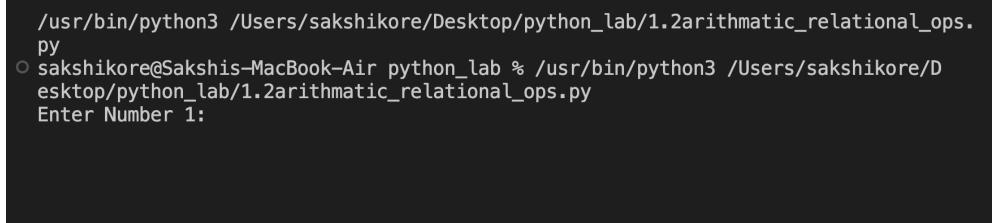
Takes user input for two numbers **x** and **y**, and performs basic arithmetic operations on them like addition, subtraction, multiplication, division, modulus, and floor division.

Code:

```
x = int(input("Enter Number 1: "))
y = int(input("Enter Number 2: "))

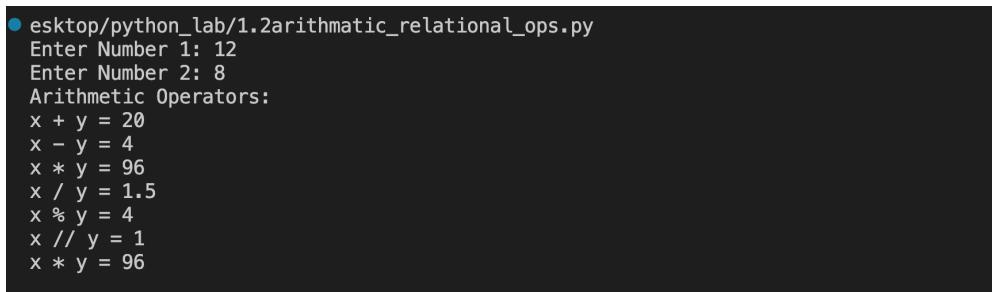
print("Arithmetic Operators:")
print("x + y =", x + y)
print("x - y =", x - y)
print("x * y =", x * y)
print("x / y =", x / y)
print("x % y =", x % y)
print("x // y =", x // y)
print("x * y =", x * y)
print()
```

Output: (screenshot)



```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.2arithmetic_relational_ops.py
sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.2arithmetic_relational_ops.py
Enter Number 1:
Enter Number 2:
```

Test Case: Any two (screenshot)



```
Desktop/python_lab/1.2arithmetic_relational_ops.py
Enter Number 1: 12
Enter Number 2: 8
Arithmetic Operators:
x + y = 20
x - y = 4
x * y = 96
x / y = 1.5
x % y = 4
x // y = 1
x * y = 96
```

Conclusion:

On taking input from user, the code prints the answers of all arithmetic operations for the entered values with respect to the formulas mentioned in the code.

Experiment No: 1.3

Title: Write a program to find whether a given no is even & odd.

Theory:

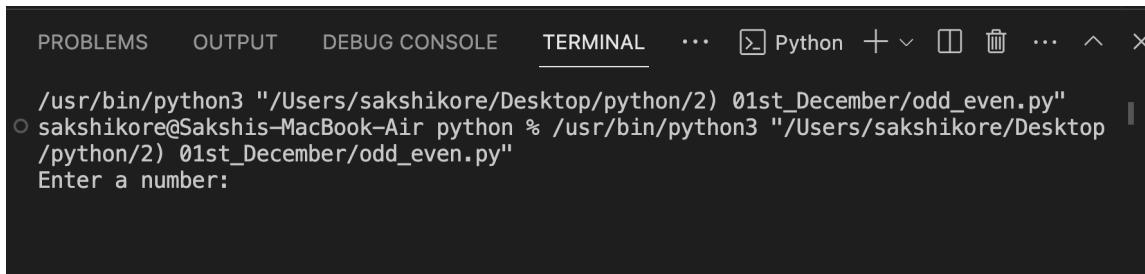
This program determines whether a given number is even or odd. It takes user input for a number, performs the modulo operation with 2, and checks if the result is equal to zero. If true, it concludes that the number is even; otherwise, it identifies the number as odd.

Code:

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

if (num % 2) == 0:
    print(num, " is a Even Number")
else:
    print(num, " is a Odd Number")
```

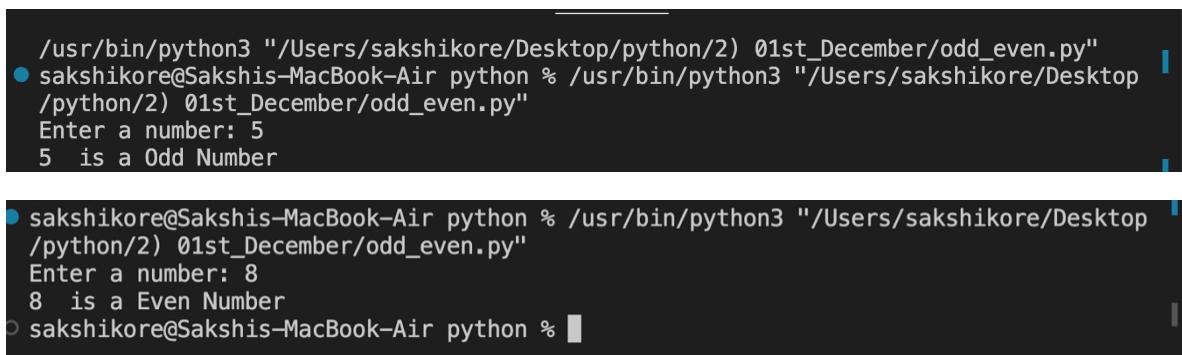
Output: (screenshot)



A screenshot of a terminal window. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and others. Below the tabs, there is some system information and a command line prompt. The main area of the terminal shows the following text:

```
/usr/bin/python3 "/Users/sakshikore/Desktop/python/2) 01st_December/odd_even.py"
sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/
/python/2) 01st_December/odd_even.py"
Enter a number:
```

Test Case: Any two (screenshot)



A screenshot of a terminal window showing two separate runs of the program. The first run shows the program being run with a number 5, and it outputs that 5 is an odd number. The second run shows the program being run with a number 8, and it outputs that 8 is an even number.

```
/usr/bin/python3 "/Users/sakshikore/Desktop/python/2) 01st_December/odd_even.py"
sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/
/python/2) 01st_December/odd_even.py"
Enter a number: 5
5 is a Odd Number

sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/
/python/2) 01st_December/odd_even.py"
Enter a number: 8
8 is a Even Number
sakshikore@Sakshis-MacBook-Air python %
```

Conclusion:

The code determines whether a user-inputted number is even or odd, with the use of the modulo operator to evaluate divisibility by 2.

Experiment No: 1.4

Title: Write a program to print first n natural number & their sum.

Theory:

The program calculates the sum of natural numbers up to a given input 'n' using a while loop. It initialises variables for sum and 'i', continuously adds 'i' to the sum until 'i' reaches the value of 'n'.

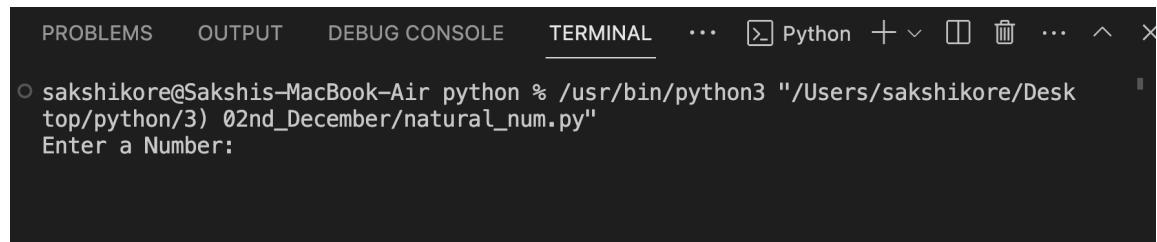
Code:

```
n = int(input ("Enter a Number: "))
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i + 1

print ("The sum is: ", sum)
```

Output: (screenshot)



Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/python/3) 02nd_December/natural_num.py"
Enter a Number: 4
The sum is: 10
 - sakshikore@sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/python/3) 02nd_December/natural_num.py"
Enter a Number: 9
The sum is: 45

Conclusion:

The program computes the sum of natural numbers up to the specified 'n' with respect to the formula mentioned in the code.

Experiment No: 1.5

Title: Write a program to determine whether the character entered is a Vowel or not

Theory:

This program determines whether a user-inputted character is a vowel or a consonant. It checks if character belongs to a predefined list of vowels, and then prints the answer.

Code:

```
char = input("Enter a character: ")  
vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']  
  
if char in vowels:  
    print(f"The character '{char}' is a vowel!")  
else:  
    print(f"The character '{char}' is a consonant!")
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    TERMINAL    ...
TERMINAL    ...
Python    +    ∨    □    🗑    ...    ^    ×
○ sakshikore@Sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/python/11) 14th_December/test.py"
Enter a character: █
```

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/python/11) 14th_December/test.py"
Enter a character: n
The character 'n' is a consonant!
- sakshikore@sakshis-MacBook-Air python % /usr/bin/python3 "/Users/sakshikore/Desktop/python/11) 14th_December/test.py"
Enter a character: a
The character 'a' is a vowel!

Conclusion:

By using a list of vowels and a simple conditional check, the program identifies whether a given character is a vowel or a consonant.

Experiment No: 1.6

Title: Write a program to find whether given number is an Armstrong Number.

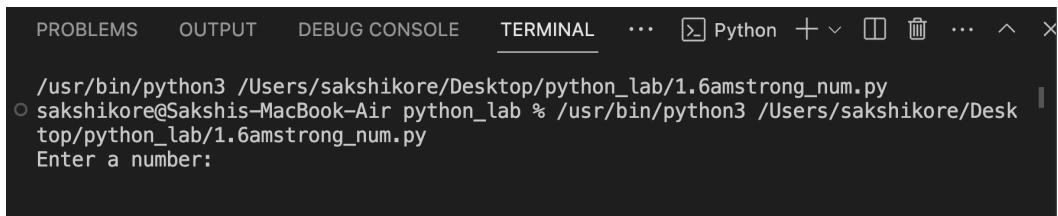
Theory:

This program takes user input, calculates the number of digits, and then computes sum of each digit raised to power of number of digits. It compares the calculated sum with the original number to check if it is an armstrong number or not

Code:

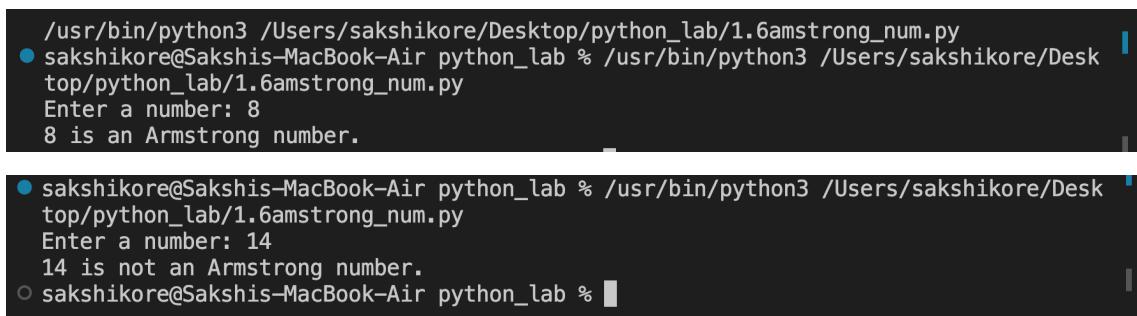
```
a = int(input("Enter a number: "))
numbers = len(str(a))
sum = 0
temp = a
while temp > 0:
    digit = temp % 10
    sum += digit ** numbers
    temp //= 10
if a == sum:
    print(f"{a} is an Armstrong number.")
else:
    print(f"{a} is not an Armstrong number.)")
```

Output: (screenshot)



A screenshot of a terminal window. The title bar shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and Python. The terminal content shows the command /usr/bin/python3 followed by the path to the script and the script name 1.6amstrong_num.py. A prompt shows the user's name and host: sakshikore@Sakshis-MacBook-Air python_lab %. The user then types 'Enter a number:' followed by a blank line.

Test Case: Any two (screenshot)



A screenshot of a terminal window showing two separate runs of the program. The first run shows the user entering the number 8, which is identified as an Armstrong number. The second run shows the user entering the number 14, which is identified as not being an Armstrong number.

Conclusion:

It defines the logic behind calculation of Armstrong number and identifies the same.

Experiment No: 1.7

Title: Write a program using for loop to calculate factorial of a No.

Theory:

This code calculates the factorial of a given number using a for loop. It takes user input, initialises a variable factorial to 1, and then multiplies it by each integer from 1 to the input number.

Code:

```
num = int(input("Enter a number: "))
factorial = 1

for i in range(1, num + 1):
    factorial *= i

print(f"The factorial of {num} is: {factorial}")
```

Output: (screenshot)

Test Case: Any two (screenshot)

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.7factorial.py
● esktop/python_lab/1.7factorial.py
Enter a number: 5
The factorial of 5 is: 120

● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.7factorial.py
Enter a number: 1
The factorial of 1 is: 1
```

Conclusion:

By using for loop, the program computes the factorial of the number with respect to the formula mentioned in the code.

Experiment No: 1.8 (i)

Title: Write a program to print right angled triangle using astrics

Theory:

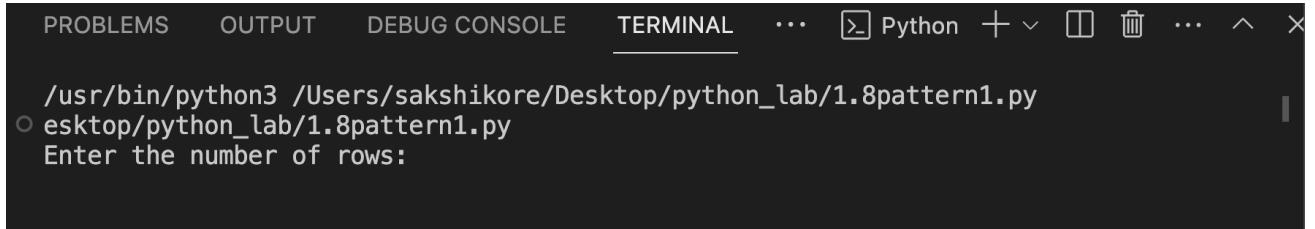
This program generates a simple pattern of asterisks in the shape of a right-angled triangle. It prompts the user to input the number of rows, and then, using nested loops, prints asterisks in incremental order on each line.

Code:

```
n = int(input('Enter the number of rows: '))

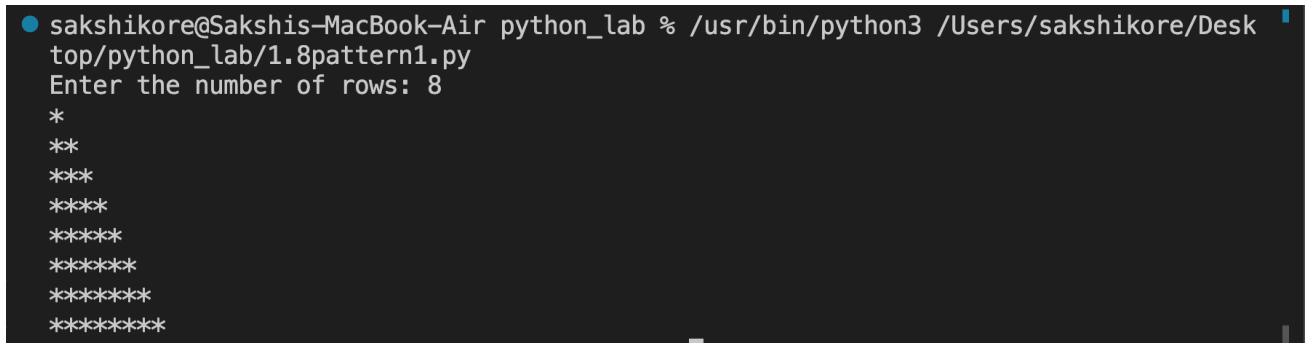
for i in range (0, n):
    for j in range (0, i + 1):
        print ('*', end = '')
    print ()
```

Output: (screenshot)



A screenshot of a terminal window. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and others. Below the tabs, the command `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.8pattern1.py` is entered. A cursor is visible at the end of the command. The terminal then prompts `Enter the number of rows:`. The user has not yet typed a response.

Test Case: Any two (screenshot)



A screenshot of a terminal window showing the execution of the Python script. The command `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.8pattern1.py` is run. The terminal then prompts `Enter the number of rows:`. The user enters `8`. The terminal then displays a right-angled triangle made of asterisks, with 8 rows. The triangle is as follows:

```
*
**
***
****
*****
******
*****
*****
```

Conclusion:

The program demonstrates a basic pattern printing technique using nested loops, creating a right-angled triangle with asterisks.

Experiment No: 1.8 (ii)

Title: Write a program to print 1-22-333-4444-55555 Pattern

Theory:

This program generates a pattern where each row displays numbers incrementally up to the row number. It takes user input for the number of rows, and using nested loops, prints the row number repetitively on each line.

Code:

```
n = int(input("Enter number of rows: "))

for i in range(1,n+1):
    for j in range(1, i+1):
        print(i, end="")
    print()
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    ⚒ Python    +    ▾    ⏺    ⏻    ...    ^    ×  
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.8pattern2.py  
○ sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.8pattern2.py  
Enter number of rows:
```

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.8pattern2.py
Enter number of rows: 8
1
22
333
4444
55555
666666
7777777
88888888

Conclusion:

By using nested loops, the program creates a simple numerical pattern, in right angle with the help of the concept of nested iteration.

Experiment No: 1.8 (iii)

Title: Write a program to print a pyramid pattern

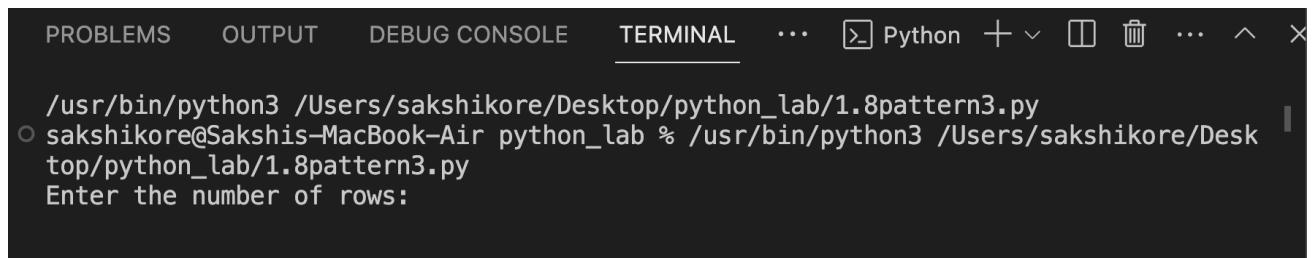
Theory:

This program generates a pattern of pyramid with asterisks. It takes user input for the number of rows, uses nested loops to control spacing and asterisk printing, creating the pyramid pattern.

Code:

```
n = int(input("Enter the number of rows: "))
m = n - 1
for i in range (0, n):
    for j in range (0, m):
        print (" ", end = " ")
    m = m - 1
    for j in range (0, i+1):
        print ("*", end = " ")
    print("\r")
```

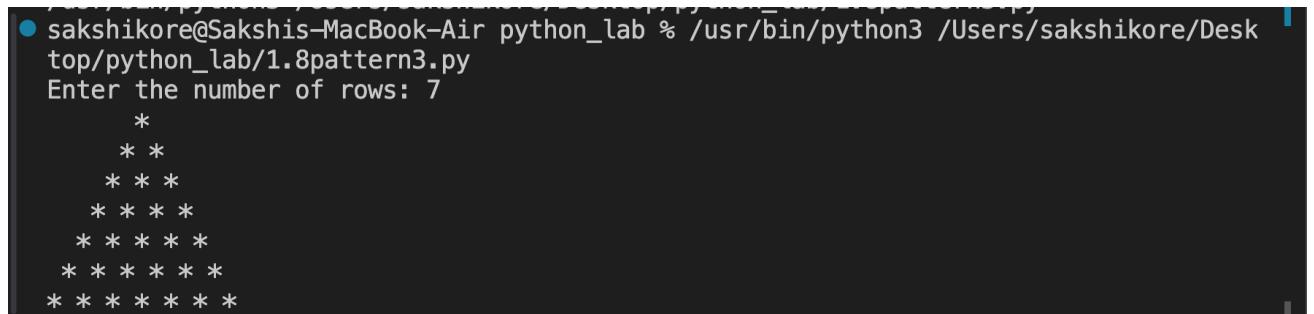
Output: (screenshot)



A screenshot of a terminal window. The title bar shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and Python. The terminal content shows the command `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/1.8pattern3.py` being run. The user is prompted to enter the number of rows, which they type as 7. The output shows the following pyramid pattern:

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Test Case: Any two (screenshot)



A screenshot of a terminal window showing the execution of the script. The user enters the number of rows as 7. The output displays a pyramid pattern with 7 rows of asterisks:

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Conclusion:

By combining loops to control spacing and print characters, the program creates a pyramid pattern with asterisks.

Experiment No: 2.1

Title: Write a program that defines the list of countries that are in BRICS.

Theory:

The program checks if a user-inputted country is a member of the BRICS group. It uses a list of BRICS countries, converts the user input to lowercase for case matching, and then check using an **if** statement and displays the result

Code:

```
countries = ["brazil", "russia", "india", "china", "srilanka" ]  
  
member = input("Enter the name of the country: ").lower()  
  
if member in countries:  
    print(member, " is the member of BRICS")  
else:  
    print(member, " is not a member of BRICS")
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    ⚙ Python    +    ▾    ⏮    ⏹    ⏷    ⏸    ⏹    ⏵
```

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.1brics.py
sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.1brics.py
Enter the name of the country:
```

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.1brics.py
Enter the name of the country: india
india is the member of BRICS

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.1brics.py
Enter the name of the country: singapore
singapore is not a member of BRICS

Conclusion:

The program determines whether a given country is a member of BRICS, showcasing the use of lists, conditional statements, and user input processing

Experiment No: 2.2

Title: Write a program to traverse a list in reverse order.

- 1.By using Reverse method.
 - 2.By using slicing

Theory:

The program takes user input to create a list of elements and then demonstrates two methods to traverse the list in reverse order: using the reversed method and by slicing.

Code:

```
mylist = input("Enter elements of the list: ").split()

print("List in reverse order using Reverse method:")
for item in reversed(mylist):
    print(item, end=" ")

print()

print("List in reverse order using Slicing:")
for item in mylist[::-1]:
    print(item, end=" ")
```

Output: (screenshot)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ... Python + ... ^ x

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.2reverse_order.py
sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.2reverse_order.py
Enter elements of the list:
```

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.2reverse_order.py
Enter elements of the list: 0 8 17 3 9 5 22
List in reverse order using Reverse method:
22 5 9 3 17 8 0
List in reverse order using Slicing:
22 5 9 3 17 8 0 %

Conclusion:

By using both the reversed and slicing method the program showcases two approaches for traversing a list in reverse order.

Experiment No: 2.3

Title: Write a program that scans the email address and forms a tuple of username and domain.

Theory:

This program takes user input of email addresses & then uses the split method to separate individual email address into username and domain parts using the '@' symbol.

Code:

```
emails = input("Enter the Email addresses separated by commas: ")
elist = emails.split(',')

for email in elist:
    username, domain = email.strip().split('@')
    print("Username: ", username, "Domain: ", domain)
```

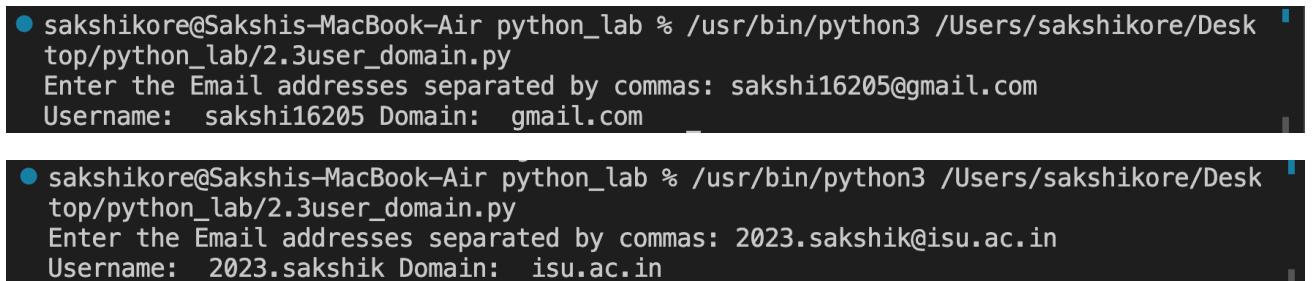
Output: (screenshot)



A screenshot of a terminal window. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and others. The terminal content shows a command-line session:

```
sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.3user_domain.py
Enter the Email addresses separated by commas: 
```

Test Case: Any two (screenshot)



Two screenshots of a terminal window showing test cases for the script:

- First test case:
Command: `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.3user_domain.py`
Input: `Enter the Email addresses separated by commas: sakshi16205@gmail.com`
Output: `Username: sakshi16205 Domain: gmail.com`
- Second test case:
Command: `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.3user_domain.py`
Input: `Enter the Email addresses separated by commas: 2023.sakshik@isu.ac.in`
Output: `Username: 2023.sakshik Domain: isu.ac.in`

Conclusion:

The program extracts and prints the username and domain for each email from a list of strings, with the use of string manipulation and the split method.

Experiment No: 2.4

Title: Write a program to create a list of tuples from given list having number and add its cube in tuple.

i/p: c= [2,3,4,5,6,7,8,9]

Theory:

The code takes user input to create a list of numbers and then uses a list comprehension to generate a list of tuples, each containing a number from the input list and its cube.

Code:

```
numbers = [int(n) for n in input("Enter numbers separated by  
spaces: ").split()]  
  
cubes_tuples = [(num, num ** 3) for num in numbers]  
  
print("List of tuples giving cube for num:", cubes_tuples)
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    ⚙ Python    +    ⌂    ⏷    ...    ⌄    ⌁  
○ /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.4cube_tuple.py  
○ sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.4cube_tuple.py  
Enter numbers separated by spaces:
```

Test Case: Any two (screenshot)

- sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.4cube_tuple.py
Enter numbers separated by spaces: 7 9 3 2 4
List of tuples giving cube for num: [(7, 343), (9, 729), (3, 27), (2, 8), (4, 64)]
 - sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.4cube_tuple.py
Enter numbers separated by spaces: 2 1 13 0
List of tuples giving cube for num: [(2, 8), (1, 1), (13, 2197), (0, 0)]

Conclusion:

By using list comprehension, the program efficiently creates a list of tuples with numbers and their respective cubes with help of formula written in the code.

Experiment No: 2.5

Title: Write a program to compare two dictionaries in Python?

(By using == operator)

Theory:

The program compares two dictionaries using the `==` operator. The dictionaries, `dict1` and `dict2`, are checked for equality, and the result is stored in the variable `equal`.

Code:

```
dict1 = {"a": 1, "b": 2, "c": 3}  
dict2 = {"a": 1, "b": 2, "c": 3}
```

```
equal = dict1 == dict2
```

```
print("Dictionaries are equal." if equal else "Dictionaries are not equal.")
```

Output: (screenshot)

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.5dict_compare.py
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.5dict_compare.py
Dictionaries are equal.
○ sakshikore@Sakshis-MacBook-Air python_lab %
```

Test Case: Any two (screenshot)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL ... Python + ⌂ ⚡ ⏪ ⏴ ⏵ ⏹

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.5dict_compare.py
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.5dict_compare.py
Dictionaries are equal.
○ sakshikore@Sakshis-MacBook-Air python_lab %
```

Conclusion:

The program uses the `==` operator to compare two dictionaries. If the dictionaries have the same key-value pairs, they are considered equal, otherwise, they are considered not equal.

Experiment No: 2.6

Title: Write a program that creates dictionary of cube of odd numbers in the range.

Theory:

The program takes user input of starting and ending numbers. It then creates a dictionary containing cubes of odd numbers within the specified range.

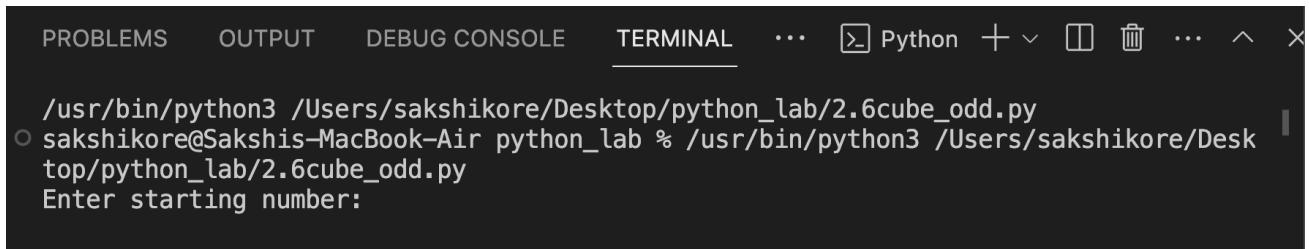
Code:

```
start = int(input("Enter starting number: "))
end = int(input("Enter ending number: "))

cube_dict = {num: num ** 3 for num in range(start, end + 1) if num
% 2 != 0}

print("Dictionary of Cubes for Odd Numbers:", cube_dict)
```

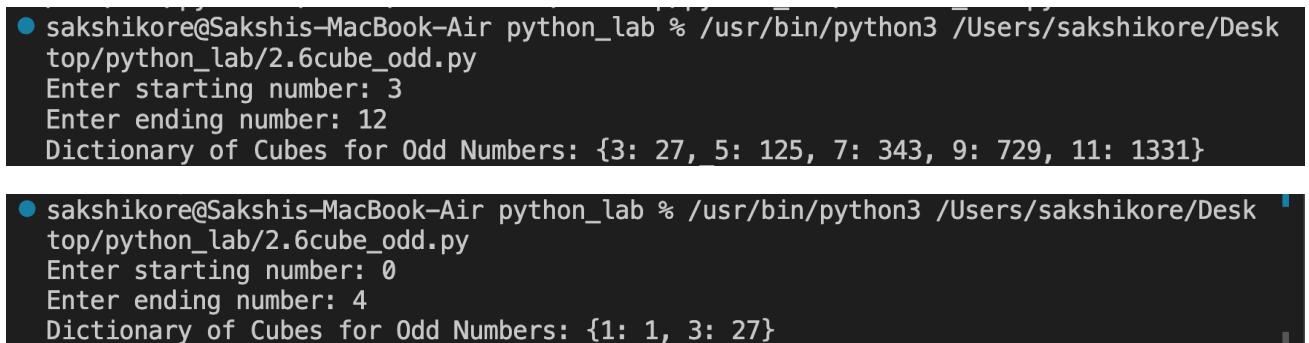
Output: (screenshot)



A screenshot of a terminal window. The window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and others. Below the tabs, there is a command line history and a prompt. The command entered is `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.6cube_odd.py`. The terminal then shows the output of the script, which asks for the starting and ending numbers, and then prints the resulting dictionary of cubes for odd numbers.

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.6cube_odd.py
sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.6cube_odd.py
Enter starting number:
```

Test Case: Any two (screenshot)



A screenshot of a terminal window showing two separate runs of the program. The first run starts with the command `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.6cube_odd.py`, followed by user input for the starting and ending numbers, and the resulting dictionary of cubes. The second run starts with the same command, followed by different user input, and also produces a dictionary of cubes.

```
sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.6cube_odd.py
Enter starting number: 3
Enter ending number: 12
Dictionary of Cubes for Odd Numbers: {3: 27, 5: 125, 7: 343, 9: 729, 11: 1331}

sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/2.6cube_odd.py
Enter starting number: 0
Enter ending number: 4
Dictionary of Cubes for Odd Numbers: {1: 1, 3: 27}
```

Conclusion:

By using dictionary comprehension, the program creates a dictionary of cubes for odd numbers in the range.

Experiment No: 2.7

Title: Write a program for various list slicing operation.

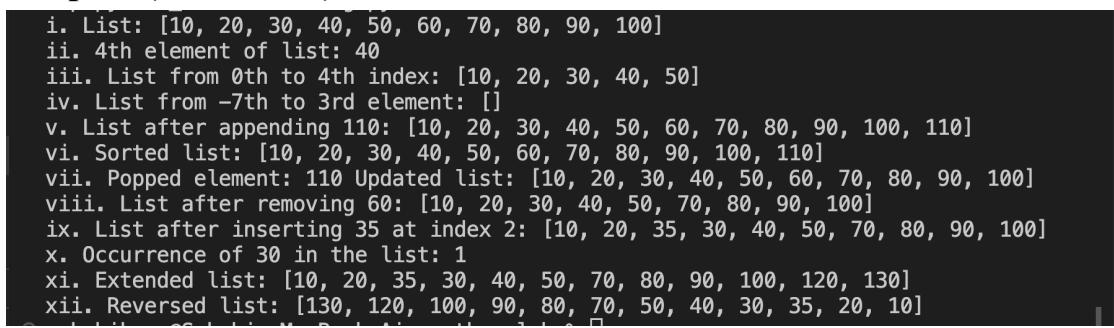
Theory:

This program uses list slicing operations on given list. It covers printing specific elements, slicing, appending, sorting, popping, removing, inserting, counting occurrences, extending, and reversing the list.

Code:

```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
print("i. List:", a)
print("ii. 4th element of list:", a[3])
print("iii. List from 0th to 4th index:", a[:5])
print("iv. List from -7th to 3rd element:", a[-7:3])
a.append(110)
print("v. List after appending 110:", a)
a.sort()
print("vi. Sorted list:", a)
popped_element = a.pop()
print("vii. Popped element:", popped_element, "Updated list:", a)
a.remove(60)
print("viii. List after removing 60:", a)
a.insert(2, 35)
print("ix. List after inserting 35 at index 2:", a)
count_30 = a.count(30)
print("x. Occurrence of 30 in the list:", count_30)
a.extend([120, 130])
print("xi. Extended list:", a)
a.reverse()
print("xii. Reversed list:", a)
```

Output: (Screenshot)



```
i. List: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
ii. 4th element of list: 40
iii. List from 0th to 4th index: [10, 20, 30, 40, 50]
iv. List from -7th to 3rd element: []
v. List after appending 110: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
vi. Sorted list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
vii. Popped element: 110 Updated list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
viii. List after removing 60: [10, 20, 30, 40, 50, 70, 80, 90, 100]
ix. List after inserting 35 at index 2: [10, 20, 35, 30, 40, 50, 70, 80, 90, 100]
x. Occurrence of 30 in the list: 1
xi. Extended list: [10, 20, 35, 30, 40, 50, 70, 80, 90, 100, 120, 130]
xii. Reversed list: [130, 120, 100, 90, 80, 70, 50, 40, 30, 35, 20, 10]
```

Conclusion:

The program shows a variety of list manipulation operations using slicing.

Experiment No: 3.1

Title: Write a program to extend a list in python by using given approach.

Theory:

This program uses three different approaches to extend a list:

- i. By using the + operator,
 - ii. By using the append() method,
 - iii. By using the extend() method.

Code:

```
list = [1, 2, 3]
```

```
# i. By using + operator  
extend1 = list + [4, 5, 6]  
print("i. Extended list using + operator:", extend1)
```

```
# i.e. By using Append()
```

```
list.append(4)
list.append(5)
list.append(6)
print("ii. Extended list using Append():", list)
```

iii. By using extend()

```
list = [1, 2, 3]
list.extend([4, 5, 6])
print("iii. Extended list using extend():", list)
```

Output: (screenshot)

/usr/bin/python3 /Users/sakshikore/Desktop/python_lab
/3.1append.py

- sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.1append.py
 - i. Extended list using + operator: [1, 2, 3, 4, 5, 6]
 - ii. Extended list using Append(): [1, 2, 3, 4, 5, 6]
 - iii. Extended list using extend(): [1, 2, 3, 4, 5, 6]
- sakshikore@Sakshis-MacBook-Air python_lab %

Conclusion:

The program uses multiple methods to extend a given list in python.

Experiment No: 3.2

Title: Write a program to add two matrices.

Theory:

This program takes user input for number of rows and columns, then asks the user to enter elements for two matrices. It performs matrix addition and displays the result.

Code:

```
rows = int(input("Enter the Number of rows : " ))
column = int(input("Enter the Number of Columns: "))
print("Enter the elements of First Matrix: ")
matrix1= [[int(input()) for i in range(column)] for i in range(rows)]
print("First Matrix is: ")
for n in matrix1:
    print(n)
print("Enter the elements of Second Matrix:")
matrix2= [[int(input()) for i in range(column)] for i in range(rows)]
for n in matrix2:
    print(n)

result=[[0 for i in range(column)] for i in range(rows)]
for i in range(rows):
    for j in range(column):
        result[i][j] = matrix1[i][j]+matrix2[i][j]
print("The Sum of Above two Matrices is : ")
for r in result:
    print(r)
```

Output: (screenshot)

Test Case: Any two (screenshot)

```
Enter the Number of rows : 2
Enter the Number of Columns: 3
Enter the elements of First Matrix:
2
6
1
3
4
8
First Matrix is:
[2, 6, 1]
[3, 4, 8]
Enter the elements of Second Matrix:
9
1
5
3
7
0
[9, 1, 5]
[3, 7, 0]
The Sum of Above two Matrices is :
[11, 7, 6]
[6, 11, 8]
```

Conclusion:

The program shows matrix addition by utilising nested lists to represent matrices.

Experiment No: 3.3

Title: Write a Python function that takes a list and returns a new list with distinct elements from the first list.

Theory:

The code defines a function `getelements` that takes input of list, goes through it and then creates a new list containing only unique elements. It does this by checking whether each element is already present in the result list before appending.

Code:

```
def getelements(list):
    distinctlist = []
    for element in list:
        if element not in distinctlist:
            distinctlist.append(element)
    return distinctlist

a = input("Enter elements for list: ")
userlist = a.split()
userlist = [int(element) for element in userlist]
result = getelements(userlist)
print("List:", userlist)
print("List with distinct elements:", result)
```

Output: (screenshot)

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.3distinct_list.py
○ sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.3distinct_list.py
Enter elements for list:
```

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.3distinct_list.py
Enter elements for list: 2 8 0 11 2 9 0 1
List: [2, 8, 0, 11, 2, 9, 0, 1]
List with distinct elements: [2, 8, 0, 11, 9, 1]

Conclusion:

By using a `for` loop, the code removes duplicate elements from the list, and creates a new list with distinct elements.

Experiment No: 3.4

Title: Write a program to Check whether a number is perfect or not.

Theory:

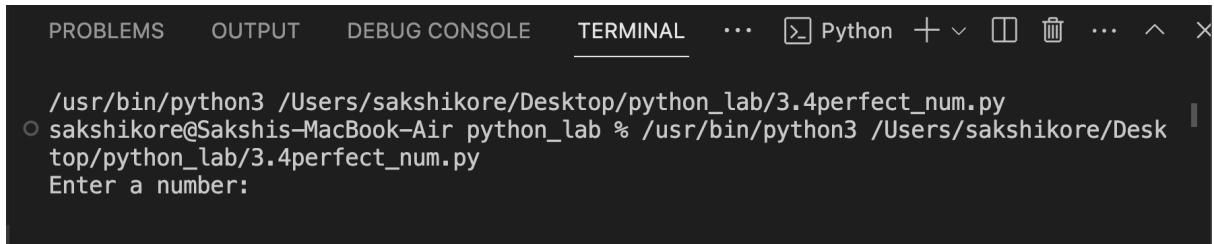
The code defines a function **perfectnum** to check if a given number is perfect. A perfect number is one whose sum of divisors (excluding itself) equals the number. The code calculates the sum and checks if it matches the input number.

Code:

```
def perfectnum(number):
    if number <= 0:
        return False
    divsum = sum([divisor for divisor in range(1, number) if number %
% divisor == 0])
    return divsum == number
check = int(input("Enter a number: "))
result = perfectnum(check)

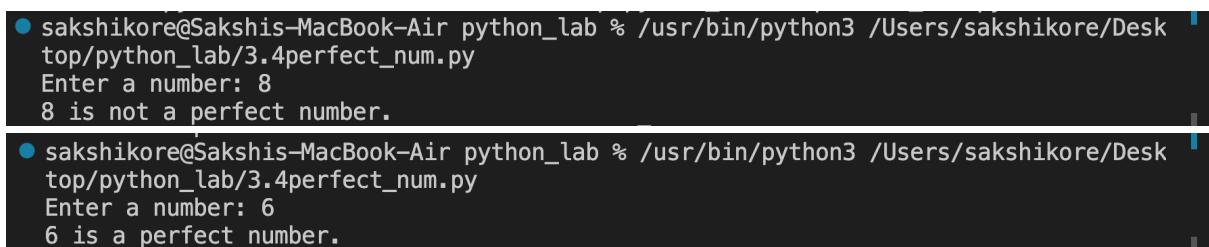
if result:
    print(f"{check} is a perfect number.")
else:
    print(f"{check} is not a perfect number.)
```

Output: (screenshot)



A screenshot of a terminal window. The title bar shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and Python. The terminal content shows the command `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.4perfect_num.py` being run. The user prompt `sakshikore@Sakshis-MacBook-Air python_lab %` is followed by the script name. Below the command, the user types `Enter a number:`. The terminal is currently empty below the prompt.

Test Case: Any two (screenshot)



A screenshot of a terminal window showing two test cases. The first case shows the user entering `8`, which is identified as `not a perfect number`. The second case shows the user entering `6`, which is identified as `a perfect number`.

```
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.4perfect_num.py
Enter a number: 8
8 is not a perfect number.

● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.4perfect_num.py
Enter a number: 6
6 is a perfect number.
```

Conclusion:

The code identifies whether the entered number is a perfect number or not, and provides the result of the evaluation.

Experiment No: 3.5

Title: Write a Python function that accepts a string and counts the number of upper- and lower-case letters.

```
string_test= 'Today is My Best Day'
```

Theory:

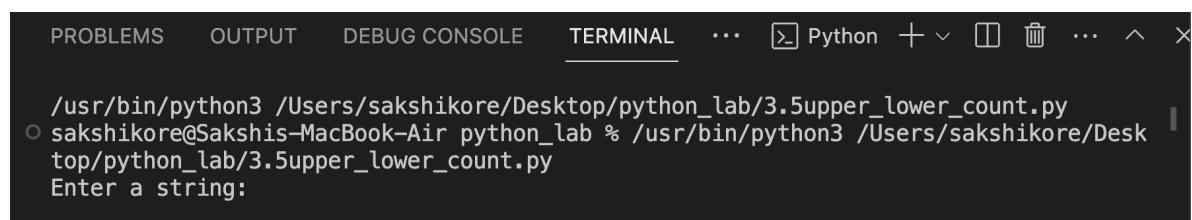
The function **count** takes a string as input and calculates the count of uppercase and lowercase letters in the string using **isupper** and **islower** string methods

Code:

```
def count(string):
    upper_count = sum(1 for char in string if char.isupper())
    lower_count = sum(1 for char in string if char.islower())
    return upper_count, lower_count

test = input("Enter a string: ")
upper, lower = count(test)
print(f"Uppercase letters: {upper}. Lowercase letters: {lower}")
```

Output: (screenshot)



Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/3.5upper_lower_count.py
Enter a string: Today is My Best Day
Uppercase letters: 4, Lowercase letters: 12

Conclusion:

On applying the function to the provided string, it counts and prints the number of uppercase and lowercase letters.

Experiment No: 4.1

Title: Write a program to Create Employee Class & add methods to get employee details & print.

Theory:

The code defines a simple **Employee** class with attributes such as employee ID, name, gender, city, and salary. The class has an `__init__` method to initialize these attributes.

Code:

```
class Employee:  
    def __init__(self, emp_id, name, gender, city, salary):  
        self.emp_id = emp_id  
        self.name = name  
        self.gender = gender  
        self.city = city  
        self.salary = salary  
  
def main():  
    emp_id = input("Enter Employee ID: ")  
    name = input("Enter Name: ")  
    gender = input("Enter Gender: ")  
    city = input("Enter City: ")  
    salary = float(input("Enter Salary: "))  
  
    employee = Employee(emp_id, name, gender, city, salary)  
  
    print("\nEmployee Details:")  
    print("ID:", employee.emp_id)  
    print("Name:", employee.name)  
    print("Gender:", employee.gender)  
    print("City:", employee.city)  
    print("Salary:", employee.salary)  
  
if __name__ == "__main__":  
    main()
```

Output: (screenshot)

Test Case: Any two (screenshot)

```
● sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.1employee.py
Enter Employee ID: 98
Enter Name: Sakshi
Enter Gender: Female
Enter City: Mumbai
Enter Salary: 40000

Employee Details:
ID: 98
Name: Sakshi
Gender: Female
City: Mumbai
Salary: 40000.0
```

Conclusion:

When executed, the code prompts the user to input details for a new employee, creates an instance of the Employee class, and displays the entered information. This structure allows for easy management and representation of employee data in a clear and organised manner.

Experiment No: 4.2

Title: Write a program to take input as name, email & age from user using combination of keywords argument and positional arguments (*args and **kwargs) using function

Theory:

The program uses a function with a combination of positional arguments (*args) and keyword arguments (**kwargs) to obtain user details for name, email, and age. The user input is collected in the main function and passed to the flexible get_user_details function for processing

Code:

```
def userdetails(*args, **kwargs):
    name = args[0] if args else kwargs.get('name', 'Unknown')
    email = kwargs.get('email', 'Unknown')
    age = kwargs.get('age', 'Unknown')

    return name, email, age

def main():
    name_input = input("Enter your name: ")
    email_input = input("Enter your email: ")
    age_input = input("Enter your age: ")

    details = userdetails(name=name_input, email=email_input,
age=age_input)

    print("\nUser Details:")
    print("Name:", details[0])
    print("Email:", details[1])
    print("Age:", details[2])

if __name__ == "__main__":
    main()
```

Output: (screenshot)

The screenshot shows a Jupyter Notebook interface with a terminal tab active. The terminal window displays the command `/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.2args_kwargs.py`. The output shows the script running and prompting the user for input. The user has typed "Enter your name: " followed by a cursor. The rest of the terminal window is blank.

Test Case: Any two (screenshot)

```
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.2args_kwargs.py
Enter your name: sakshi
Enter your email: sakshikore16@gmail.com
Enter your age: 19

User Details:
Name: sakshi
Email: sakshikore16@gmail.com
Age: 19
```

Conclusion:

By leveraging both positional and keyword arguments, this program allows users to input their details conveniently, providing a flexible and readable way to handle varying sets of information. The result is a clear display of the user's name, email, and age.

Experiment No: 4.3

Title: Write a program to admit the students in the different Departments(pgdm btech)and count the students. (Class, Object and Constructor).

Theory:

The Python code defines a Student class representing student details. The user is prompted to input information for multiple students & categorizes them into PGDM or B.Tech departments.

Code:

```
class Student:
    count = 0

    def __init__(self):
        self.name = input("Enter Student Name: ")
        self.age = int(input("Enter Student Age: "))
        self.department = input("Enter Student Department (PGDM(p)/
B.Tech(b)): ").capitalize()

        Student.count += 1
    def display(self):
        print("Name:", self.name, "Age:", self.age, "Department:",
self.department)
print("----- STUDENT ADMIT -----")
pgdm_students = []
btech_students = []
num_students = int(input("Enter The Total Number Of Students: "))
for _ in range(num_students):
    new_student = Student()
    new_student.display()

    if new_student.department == 'P':
        pgdm_students.append(new_student)

    elif new_student.department == 'B':
        btech_students.append(new_student)
print("*****")
print("\nTotal PGDM Department Students:")
for student in pgdm_students:
    student.display()
print("\nTotal B.Tech Department Students:")
for student in btech_students:
    student.display()
print("\nTotal Number Of students:", Student.count)
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    ⚡ Python    +    □    ⏮    ⏹    ⏷    ⏸    ⏹
```

○ sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.3pgdm_btech.py
----- STUDENT ADMIT -----
Enter The Total Number Of Students:

Test Case: Any two (screenshot)

```
ech.py
----- STUDENT ADMIT -----
Enter The Total Number Of Students: 6
Enter Student Name: Sakshi Kore
Enter Student Age: 18
Enter Student Department (PGDM(p)/B.Tech(b)): b
Name: Sakshi Kore Age: 18 Department: B
Enter Student Name: Arpita Jadhav
Enter Student Age: 22
Enter Student Department (PGDM(p)/B.Tech(b)): p
Name: Arpita Jadhav Age: 22 Department: P
Enter Student Name: Yashika Thakur
Enter Student Age: 18
Enter Student Department (PGDM(p)/B.Tech(b)): b
Name: Yashika Thakur Age: 18 Department: B
Enter Student Name: Gaurang Jadhav
Enter Student Age: 19
Enter Student Department (PGDM(p)/B.Tech(b)): b
Name: Gaurang Jadhav Age: 19 Department: B
Enter Student Name: Chandan Dumale
Enter Student Age: 23
Enter Student Department (PGDM(p)/B.Tech(b)): p
Name: Chandan Dumale Age: 23 Department: P
Enter Student Name: Tanmay Gharat
Enter Student Age: 23
Enter Student Department (PGDM(p)/B.Tech(b)): p
Name: Tanmay Gharat Age: 23 Department: P
*****
Total PGDM Department Students:
Name: Arpita Jadhav Age: 22 Department: P
Name: Chandan Dumale Age: 23 Department: P
Name: Tanmay Gharat Age: 23 Department: P

Total B.Tech Department Students:
Name: Sakshi Kore Age: 18 Department: B
Name: Yashika Thakur Age: 18 Department: B
Name: Gaurang Jadhav Age: 19 Department: B

Total Number Of students: 6
```

Conclusion:

On running the code, it collects student information, categorizes them by department, and presents detailed statistics, including the total number of students in each department and in total.

Experiment No: 4.4

Title: Write a program that has a class store which keeps the record of code and price of product display the menu of all product and prompt to enter the quantity of each item required and finally generate the bill and display the total amount.

Theory:

The Python code defines a **Store** class to simulate a store's inventory system. It allows adding products, displaying the menu, and generating a bill based on user input for product codes and quantities.

Code:

```
class Store:
    def __init__(self):
        self.products = {}
    def add_product(self, code, name, price):
        self.products[code] = {'name': name, 'price': price}
    def display_menu(self):
        print("Menu:")
        print("Code\tName\tPrice")
        for code, product in self.products.items():
            print(f"{code}\t{product['name']}\t{product['price']}")
    def generate_bill(self, order):
        total_amount = 0
        print("\n----- RECEIPT -----")
        print("Code\tName\tPrice\tQuantity\tTotal")
        for code, quantity in order.items():
            product = self.products[code]
            item_total = quantity * product['price']
            total_amount += item_total
            print(f"{code}\t{product['name']}\t{product['price']}\t{quantity}\t{item_total}")
        print("\nTotal Amount: ₹{:.2f}\n".format(total_amount))
    def main():
        store = Store()
        store.add_product('001', 'Bread', 25.00)
        store.add_product('002', 'Chips', 15.00)
        store.add_product('003', 'KitKat', 10.00)
        store.add_product('004', 'Coke', 20.00)
        store.add_product('005', 'Biscuit', 12.00)
        store.add_product('006', 'Butter', 35.00)
        store.add_product('007', 'Rice', 30.00)
        store.add_product('008', 'Lentils', 35.00)
        store.add_product('009', 'Suji', 40.00)
        store.add_product('010', 'Spice', 20.00)
        store.display_menu()
        order = {}
        while True:
            code = input("Enter the product code (or 'done' to finish): ")
            if code.lower() == 'done':
                break
            elif code in store.products:
                quantity = int(input(f"Enter the quantity for {store.products[code]['name']}:"))
                order[code] = quantity
            else:
                print("Invalid product code. Please enter a valid code.")
        store.generate_bill(order)
if __name__ == "__main__":
    main()
```

Output: (screenshot)

```
sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.4store.py
Menu:
Code    Name    Price
001    Bread   ₹25.0
002    Chips   ₹15.0
003    KitKat  ₹10.0
004    Coke    ₹20.0
005    Biscuit ₹12.0
006    Butter  ₹35.0
007    Rice    ₹30.0
008    Lentils ₹35.0
009    Suji    ₹40.0
010    Spice   ₹20.0
Enter the product code (or 'done' to finish):
```

Test Case: Any two (screenshot)

```
sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.4store.py
Menu:
Code    Name    Price
001    Bread   ₹25.0
002    Chips   ₹15.0
003    KitKat  ₹10.0
004    Coke    ₹20.0
005    Biscuit ₹12.0
006    Butter  ₹35.0
007    Rice    ₹30.0
008    Lentils ₹35.0
009    Suji    ₹40.0
010    Spice   ₹20.0
Enter the product code (or 'done' to finish): 003
Enter the quantity for KitKat: 3
Enter the product code (or 'done' to finish): 001
Enter the quantity for Bread: 1
Enter the product code (or 'done' to finish): 006
Enter the quantity for Butter: 1
Enter the product code (or 'done' to finish): done
----- RECEIPT -----
Code    Name    Price  Quantity   Total
003    KitKat  ₹10.0  3        ₹30.0
001    Bread   ₹25.0  1        ₹25.0
006    Butter  ₹35.0  1        ₹35.0
Total Amount: ₹90.00
```

Conclusion:

The program initializes a store with predefined products, displays the menu, and prompts the user to input product codes and quantities for their order. It then generates a bill with a clear receipt.

Experiment No: 4.5

Title: Write a program to take input from user for addition of two numbers using (single inheritance)

Theory:

The code defines a class `Addition` with a method `add` to perform addition. Another class `Values` inherits from `Addition` and includes a method `get_input` to collect two numbers from the user. The program then creates an instance of `Values`, gets user input, performs addition, and prints the result.

Code:

```
class addition:  
    def add(self, a, b):  
        return a + b  
class values(addition):  
    def get_input(self):  
        num1 = int(input("Enter the first number: "))  
        num2 = int(input("Enter the second number: "))  
        return num1, num2  
add_values = values()  
numbers = add_values.get_input()  
result = add_values.add(*numbers)  
print(f"The sum of {numbers[0]} and {numbers[1]} is: {result}")
```

Output: (screenshot)

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.5add_inheritance.py
Enter the first number: 6
Enter the second number: 5
The sum of 6 and 5 is: 11
 - sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.5add_inheritance.py
Enter the first number: 3
Enter the second number: -2
The sum of 3 and -2 is: 1

Conclusion:

The code uses a class hierarchy where **values** inherits the addition functionality from the base class **addition**. It prompts the user for two numbers, adds them using the inherited method, and displays the result.

Experiment No: 4.6

Title: Write a program to create two base classes LU and ITM and one derived class. (Multiple inheritance).

Theory:

The code defines classes LetsUpgrade and ITM, each representing their courses. The CourseSelector class inherits from both LetsUpgrade and ITM. It allows users to view available subjects, get details about a specific subject, and enrol in a course based on user input and eligibility criteria.

Code:

```
class LetsUpgrade:
    lu_courses=[{'Subject': 'Maths', 'Trainer': 'Saikiran', 'Duration': 100},
                {'Subject': 'Python', 'Trainer': 'Saikiran', 'Duration': 150},
                {'Subject': 'Web design', 'Trainer': 'Prasad', 'Duration': 130}]
class ITM:
    itm_courses=[{'Subject': 'Maths', 'Trainer': 'Sheetal', 'Duration': 90},
                 {'Subject': 'DSA', 'Trainer': 'Sumit', 'Duration': 200},
                 {'Subject': 'Computer Fundamentals', 'Trainer': 'Sumit',
'Duration': 150}]
class CourseSelector(LetsUpgrade, ITM):
    def print_subjects(self, selected_class):
        if selected_class == 'LetsUpgrade':
            subjects = [course['Subject'] for course in self.lu_courses]
        elif selected_class == 'ITM':
            subjects = [course['Subject'] for course in self.itm_courses]
        else:
            subjects = []
        if not subjects:
            print(f"No subjects available for {selected_class}")
            return
        print(f"Available subjects for {selected_class}: {subjects}")
        selected_subject = input("Enter the subject you want details for: ")
        if selected_subject in subjects:
            if selected_class == 'LetsUpgrade':
                selected_course = next(course for course in self.lu_courses if
course['Subject'] == selected_subject)
            elif selected_class == 'ITM':
                selected_course = next(course for course in self.itm_courses if
course['Subject'] == selected_subject)
            print(f"\nDetails of {selected_subject} in {selected_class}:")
            print(f"Trainer: {selected_course['Trainer']} sir")
            print(f"Duration: {selected_course['Duration']} hours")
        else:
            print(selected_subject, " is not available in ", selected_class)
    enroll_option = input("\nDo you wish to enroll? (yes/no): ").lower()

    if enroll_option == 'yes':
        name = input("Enter your Name: ")
        dob = input("Enter your DOB (DD/MM/YYYY): ")
        age = int(input("Enter your Age: "))
        marks = int(input("Enter your 12th Marks: "))
        location = input("Enter your Location: ")
```

```
if marks >= 60:
    print("\nCongratulations! You are enrolled in ",
selected_subject)
        print("\n----- Student Details -----")
        print("Name: ", name)
        print("Age: ", age)
        print("Date Of Birth: ", dob)
        print("12th Marks: ", marks, "%")
        print("Location: ", location)
    else:
        print("\nSorry! You are not eligible for this course.")
else:
    print(selected_subject, " is not available in ", selected_class)
course_selector = CourseSelector()
selected_class = input("Enter the class (LetsUpgrade or ITM): ")
course_selector.print_subjects(selected_class)
```

Output: (screenshot)

/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.6multiple_inheritance.py

○ sakshikore@Sakshi's-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.6multiple_inheritance.py

Enter the class (LetsUpgrade or ITM):

Test Case: Any two (screenshot)

```
● sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.6multiple_inheritance.py
Enter the class (LetsUpgrade or ITM): LetsUpgrade
Available subjects for LetsUpgrade: ['Maths', 'Python', 'Web design']
Enter the subject you want details for: Python

Details of Python in LetsUpgrade:
Trainer: Saikiran sir
Duration: 150 hours

Do you wish to enroll? (yes/no): yes
Enter your Name: Sakshi Kore
Enter your DOB (DD/MM/YYYY): 16/09/2005
Enter your Age: 18
Enter your 12th Marks: 74
Enter your Location: Mumbai

Congratulations! You are enrolled in Python

----- Student Details -----
Name: Sakshi Kore
Age: 18
Date Of Birth: 16/09/2005
12th Marks: 74 %
Location: Mumbai
```

Conclusion:

The program provides a structured way for users to explore and enroll in courses offered by LetsUpgrade or ITM. It incorporates inheritance to reuse course information from the respective classes, demonstrates user interaction for course selection, and evaluates eligibility criteria for enrollment.

Experiment No: 4.7

Title: Write a program to implement Multilevel inheritance, Grandfather→Father→Child to show property inheritance from grandfather to child.

Theory:

The code establishes a class hierarchy with Grandfather, Father, and Child classes representing successive generations. Each class has attributes related to assets, business, and education.

Code:

```
class Grandfather:
    def __init__(self, assets):
        self.assets = assets
class Father(Grandfather):
    def __init__(self, assets, business):
        super().__init__(assets)
        self.business = business
class Child(Father):
    def __init__(self, assets, business, education):
        super().__init__(assets, business)
        self.education = education
grandfather_assets = 1000
father_assets = 5000
business_info = "Family Business"
child_education = "Computer Science Engineer"
#instance
grandfather = Grandfather(assets=grandfather_assets)
father = Father(assets=father_assets, business=business_info)
child = Child(assets=None, business=None, education=None)
child.assets = grandfather.assets + father.assets
child.business = father.business
child.education = child_education
print(f"\nGrandfather's assets: ₹{grandfather.assets}")
print(f"Father's assets: ₹{father.assets}")
print(f"Child's assets: ₹{child.assets}")
print(f"\nChild's business: {child.business}")
print(f"\nChild's education: {child.education}\n")
```

Output: (screenshot)

```
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.7inheritance.py

Grandfather's assets: ₹1000
Father's assets: ₹5000
Child's assets: ₹6000

Child's business: Family Business

Child's education: Computer Science Engineer
```

Conclusion:

The program creates instances of the classes, initializing and inheriting attributes from Grandfather to Father, and to Child. It demonstrates inheritance, encapsulation, & method overriding.

Experiment No: 4.8

Title: Write a program Design the Library catalogue system using inheritance take base class (library item) and derived class (Book, DVD & Journal) Each derived class should have unique attribute and methods and system should support Check in and check out the system. (Using Inheritance and Method overriding)

Theory:

The code establishes a library management system with classes like LibraryItem, Book, DVD, and Journal. Each class models a type of library item with specific attributes.

Code:

```
class LibraryItem:
    def __init__(self, title, author, item_id, copies_sold):
        self.title = title
        self.author = author
        self.item_id = item_id
        self.copies_sold = copies_sold
        self.availability = True
    def display_info(self):
        print(f"{self.item_id}. {self.title} by {self.author} ({'Available' if
self.availability else 'Not Available'})")
class Book(LibraryItem):
    def __init__(self, title, author, item_id, genre, copies_sold):
        super().__init__(title, author, item_id, copies_sold)
        self.genre = genre
    def display_info(self):
        super().display_info()
        print(f"    Genre: {self.genre}")
class DVD(LibraryItem):
    def __init__(self, title, director, item_id, duration, copies_sold):
        super().__init__(title, director, item_id, copies_sold)
        self.director = director
        self.duration = duration
    def display_info(self):
        super().display_info()
        print(f"    Director: {self.director}, Duration: {self.duration}
minutes")
class Journal(LibraryItem):
    def __init__(self, title, author, item_id, volume, copies_sold):
        super().__init__(title, author, item_id, copies_sold)
        self.volume = volume
    def display_info(self):
        super().display_info()
        print(f"    Volume: {self.volume}")
def display_catalog(items):
    print("\nLibrary Catalogue:")
    for item in items:
        item.display_info()
books = [
    Book("The Catcher in the Rye", "J.D. Salinger", "B001", "Fiction", 100),
    Book("To Kill a Mockingbird", "Harper Lee", "B002", "Classic", 150),
    Book("The Hobbit", "J.R.R. Tolkien", "B003", "Fantasy", 120),
]
```

```

dvds = [
    DVD("Inception", "Christopher Nolan", "D001", 148, 200),
    DVD("The Shawshank Redemption", "Frank Darabont", "D002", 142, 180),
    DVD("The Dark Knight", "Christopher Nolan", "D003", 152, 220),
]
journals = [
    Journal("Nature", "Various", "J001", "Vol. 587", 50),
    Journal("Science", "Various", "J002", "Vol. 374", 60),
]
while True:
    print("\nChoose an option:")
    print("1. View Books")
    print("2. View DVDs")
    print("3. View Journals")
    print("4. Exit")
    choice = input("Enter your choice (1-4): ")
    if choice == "1":
        display_catalog(books)
    elif choice == "2":
        display_catalog(dvds)
    elif choice == "3":
        display_catalog(journals)
    elif choice == "4":
        print("Exiting program. Thank you!")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 4.")
item_id = input("Enter the item ID you want to borrow (or '0' to go back): ")
if item_id == "0":
    continue
quantity = int(input("Enter the quantity you want to borrow:"))

selected_item = None
catalog = books + dvds + journals
for item in catalog:
    if item.item_id == item_id:
        selected_item = item
        break
if selected_item and selected_item.availability and quantity <= selected_item.copies_sold:
    selected_item.availability = False
    print(f"\n{quantity} {selected_item.title}(s) successfully borrowed.")
else:
    print("\nInvalid selection or not enough copies available. Please try again.")

```

Output: (screenshot)

The screenshot shows a terminal window with the following details:

- Terminal Title:** Python
- Content:**

```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ...
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.8library.py
○ sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.8library.py

Choose an option:
1. View Books
2. View DVDs
3. View Journals
4. Exit
Enter your choice (1-4):

```

Test Case: Any two (screenshot)

```
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/4.8library.py

Choose an option:
1. View Books
2. View DVDs
3. View Journals
4. Exit
Enter your choice (1-4): 2

Library Catalogue:
D001. Inception by Christopher Nolan (Available)
    Director: Christopher Nolan, Duration: 148 minutes
D002. The Shawshank Redemption by Frank Darabont (Available)
    Director: Frank Darabont, Duration: 142 minutes
D003. The Dark Knight by Christopher Nolan (Available)
    Director: Christopher Nolan, Duration: 152 minutes
Enter the item ID you want to borrow (or '0' to go back): D002
Enter the quantity you want to borrow: 3

3 The Shawshank Redemption(s) successfully borrowed.

Choose an option:
1. View Books
2. View DVDs
3. View Journals
4. Exit
Enter your choice (1-4): 4
Exiting program. Thank you!
```

Conclusion:

The program provides a user interface to view library items by category and borrow them if available. It utilizes inheritance to represent different types of library items and maintains availability status. The program offers a practical example of object-oriented programming, encapsulation, and user interaction in a library context.

Experiment No: 5.1

Title: Write a program to create my_module for addition of two numbers and import it in main script.

Theory:

The Python program consists of two files - `my_module.py` defining a module with an `add_numbers` function for addition, and `main_file.py` importing and utilizing the module to perform user-input addition. This showcases the creation and use of a simple custom module in Python.

Code:

```
my_module.py:  
def add_numbers(a, b):  
    return a + b  
  
main_file.py:  
import my_module  
num1 = float(input("Enter first number: "))  
num2 = float(input("Enter second number: "))  
result = my_module.add_numbers(num1, num2)  
print(f"The sum of {num1} and {num2} is: {result}")
```

Output: (screenshot)

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.1addition/main_file.py
○ sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.1addition/main_file.py
Enter first number:
```

Test Case: Any two (screenshot)

- sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.1addition/main_file.py
Enter first number: 7
Enter second number: 5
The sum of 7.0 and 5.0 is: 12.0

Conclusion:

Upon execution, the main script interacts with the custom module to add two numbers entered by the user, demonstrating the modular organization of code for better reusability and maintainability in Python.

Experiment No: 5.2

Title: Write a program to create the Bank Module to perform the operations such as Check the Balance, withdraw and deposit the money in bank account and import the module in main file.

Theory:

The Python program includes a module named bank_module with a BankAccount class for performing basic banking operations. The main script imports the module, creates a bank account, and interacts with it to check balance, withdraw, and deposit money, demonstrating modularity and encapsulation in programming

Code:

```
bank_module.py
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance
    def check_balance(self):
        return self.balance
    def deposit(self, amount):
        self.balance += amount
        return f"Deposited ₹{amount}. New balance: ₹
{self.balance}"
    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            return f"Withdrew ₹{amount}. New balance: ₹
{self.balance}"
        else:
            return "Insufficient funds. Withdrawal denied."
```

```
main_script.py
from bank_module import BankAccount
account_holder_name = input("Enter account holder's name: ")
initial_balance = float(input("Enter initial balance: "))
account = BankAccount(account_holder_name, initial_balance)
print(f"\nAccount Holder: {account.account_holder}")
print("Initial Balance:", account.check_balance())
withdraw_amount = float(input("Enter the withdrawal amount: "))
print(account.withdraw(withdraw_amount))
deposit_amount = float(input("Enter the deposit amount: "))
print(account.deposit(deposit_amount))
print("Updated Balance:", account.check_balance())
```

Output: (screenshot)

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.2bank/main_script.py
○ sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.2bank/main_script.py
Enter account holder's name:
```

Test Case: Any two (screenshot)

```
● sakshikore@sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.2bank/main_script.py
Enter account holder's name: Sakshi Kore
Enter initial balance: 5000

Account Holder: Sakshi Kore
Initial Balance: 5000.0
Enter the withdrawal amount: 1500
Withdrew ₹1500.0. New balance: ₹3500.0
Enter the deposit amount: 300
Deposited ₹300.0. New balance: ₹3800.0
Updated Balance: 3800.0
○ sakshikore@sakshis-MacBook-Air python_lab %
```

Conclusion:

The code exhibits a modular approach to banking operations, encapsulating functionality within a class. It showcases the use of custom modules for better code organization and reusability, providing a structured way to manage bank account-related tasks in Python.

Experiment No: 5.3

Title: Write a program to create a package with name cars and add different modules (such as BMW, AUDI, NISSAN) having classes and functionality and import them in main file cars.

Theory:

The code demonstrates the creation of a Python package named **cars** with modules (**BMW**, **AUDI**, **NISSAN**), each defining a class representing different car models. The main script imports these modules, creates instances of the car classes, and displays information about each car.

Code:

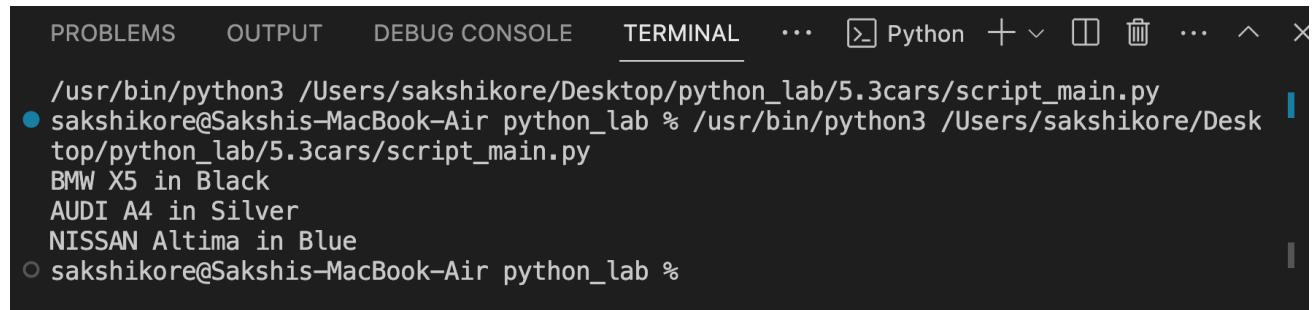
```
BMW.py
class BMW:
    def __init__(self, model, color):
        self.model = model
        self.color = color
    def display_info(self):
        print(f"BMW {self.model} in {self.color}")

AUDI.py
class AUDI:
    def __init__(self, model, color):
        self.model = model
        self.color = color
    def display_info(self):
        print(f"AUDI {self.model} in {self.color}")

NISSAN.py
class NISSAN:
    def __init__(self, model, color):
        self.model = model
        self.color = color
    def display_info(self):
        print(f"NISSAN {self.model} in {self.color}")

script_main.py
from cars import BMW, AUDI, NISSAN
bmw_car = BMW.BMW(model="X5", color="Black")
audi_car = AUDI.AUDI(model="A4", color="Silver")
nissan_car = NISSAN.NISSAN(model="Altima", color="Blue")
bmw_car.display_info()
audi_car.display_info()
nissan_car.display_info()
```

Output: (screenshot)



A screenshot of a terminal window titled "TERMINAL". The window shows the command "/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.3cars/script_main.py" being run. The output displays three car instances: "BMW X5 in Black", "AUDI A4 in Silver", and "NISSAN Altima in Blue". The terminal interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, along with standard window control buttons.

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.3cars/script_main.py
● sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/5.3cars/script_main.py
BMW X5 in Black
AUDI A4 in Silver
NISSAN Altima in Blue
○ sakshikore@Sakshis-MacBook-Air python_lab %
```

Conclusion:

The program highlights the benefits of organizing code into packages and modules, enhancing readability and maintainability. The separation of concerns allows for a structured representation of different car classes and their functionalities, providing a clear and modular design.

Experiment No: 6.1

Title: Write a program to implement Multithreading. Printing “Hello” with one thread & printing “Hi” with another thread.

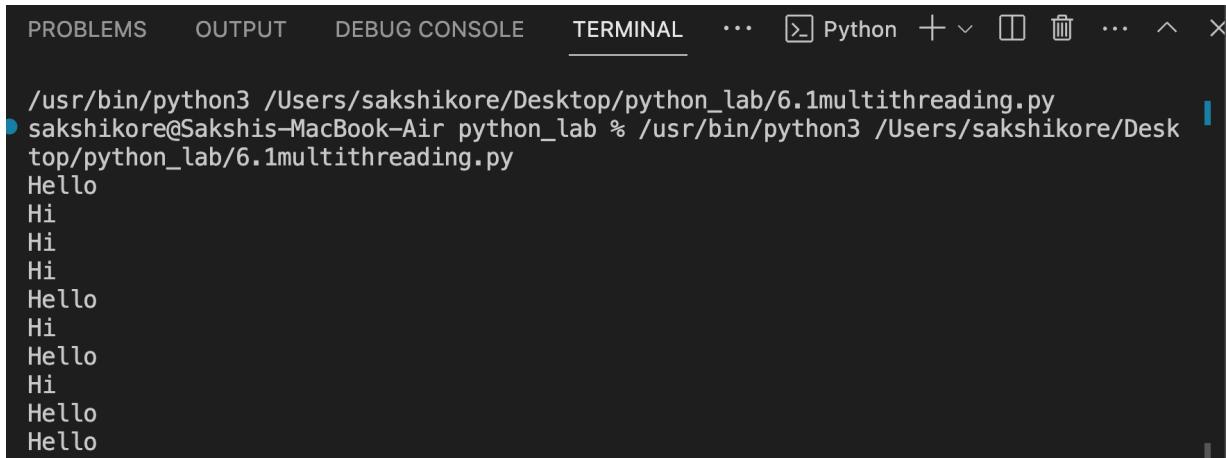
Theory:

The program demonstrates multithreading with two threads, each printing "Hello" and "Hi" in a loop concurrently. The threading module is used to create and manage threads, showcasing parallel execution of tasks.

Code:

```
import threading
def print_hello():
    for _ in range(5):
        print("Hello")
def print_hi():
    for _ in range(5):
        print("Hi")
thread_hello = threading.Thread(target=print_hello)
thread_hi = threading.Thread(target=print_hi)
thread_hello.start()
thread_hi.start()
thread_hello.join()
thread_hi.join()
```

Output: (screenshot)



A screenshot of a terminal window titled "Python". The window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and others. The TERMINAL tab is active. The terminal shows the command "/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/6.1multithreading.py" being run. The output consists of ten "Hello" and ten "Hi" messages, interleaved, demonstrating concurrent execution. The terminal interface includes standard navigation keys like arrow keys, backspace, and delete.

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/6.1multithreading.py
sakshikore@Sakhis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/6.1multithreading.py
Hello
Hi
Hi
Hi
Hello
Hi
Hello
Hi
Hello
Hello
```

Conclusion:

The code illustrates the simultaneous execution of "Hello" and "Hi" messages in an interleaved manner, highlighting the concurrent nature of multithreading. This example exemplifies the use of threads for parallel execution, enhancing program efficiency by utilizing multiple execution paths.

Experiment No: 7.1

Title: Write a program to use ‘whether API’ and print temperature of any city, also print the sunrise and sunset times for the same humidity of that area.

Theory:

This program utilizes the OpenWeather API to display current weather details for a given city. It uses the **requests** library to make an API request, fetches temperature information, converts temperatures from Kelvin to Celsius and Fahrenheit, and provides details like humidity, general weather description, sunrise, and sunset times

Code:

```
import datetime as dt
import requests
base_url = "https://api.openweathermap.org/data/2.5/weather?"
api_key = "412c6fc197bba66aadf429e7e1a835f2"
city = input("Enter City Name: ")
def kel_to_cel_fahren(kelvin):
    celsius = kelvin - 273
    fahrenheit = celsius * (9/5) + 32
    return celsius, fahrenheit
url = base_url + 'appid=' + api_key + '&q=' + city
response=requests.get(url).json()
temp_kelvin=response['main']['temp']
temp_celsius, temp_fahrenheit=kel_to_cel_fahren(temp_kelvin)
max_temp=response['main']['temp_max']
tempc,tempf=kel_to_cel_fahren(max_temp)
min_temp=response['main']['temp_min']
temc,temf=kel_to_cel_fahren(min_temp)
humidity=response['main']['humidity']
description=response['weather'][0]['description']
sunrise=dt.datetime.utcfromtimestamp(response['sys']['sunrise']+response['timezone'])
sunset=dt.datetime.utcfromtimestamp(response['sys']['sunset']+response['timezone'])
print(f"\nWeather Details For {city}")
print(f"\nTemperature: {temp_celsius:.2f}'C or {temp_fahrenheit}'F")
print(f"Maximum Temperature: {tempc:.2f}'C or {tempf}'F")
print(f"Minimum Temperature: {temc:.2f}'C or {temf}'F")
print(f"Humidity in {city}: {humidity}%")
print(f"General Weather in {city}: {description}")
print(f"Sunrises in {city} at {sunrise}.")
print(f"Sunsets in {city} at {sunset}.\n")
```

Output: (screenshot)

Test Case: Any two (screenshot)

```
WARNING:WEATHER
Enter City Name: Mumbai

Weather Details For Mumbai

Temperature: 28.14'C or 82.65199999999999'F
Maximum Temperature: 28.14'C or 82.65199999999999'F
Minimum Temperature: 28.14'C or 82.65199999999999'F
Humidity in Mumbai: 57%
General Weather in Mumbai: haze
Sunrises in Mumbai at 2024-01-01 07:11:31.
Sunsets in Mumbai at 2024-01-01 18:11:43.
```

```
Enter City Name: New Delhi
Weather Details For New Delhi
Tempertature: 11.24'C or 52.23200000000001'F
Maximum Tempertature: 11.24'C or 52.23200000000001'F
Minimum Tempertature: 11.24'C or 52.23200000000001'F
Humidity in New Delhi: 82%
General Weather in New Delhi: mist
Sunrises in New Delhi at 2024-01-01 07:13:35.
Sunsets in New Delhi at 2024-01-01 17:34:34.
```

Conclusion:

The program prompts the user to enter a city, fetches real-time weather data using the OpenWeatherMap API, and displays temperature information, humidity, weather description, sunrise, and sunset times. It showcases practical usage of APIs and data processing to provide valuable weather details for a specified location.

Experiment No: 7.2

Title: Write a program to use the ‘API’ of crypto currency.

Theory:

This Python script utilizes the CoinGecko API to retrieve and display details of a specified cryptocurrency, including its name, symbol, and current price. It uses the **requests** library to make an API request to the CoinGecko endpoint for cryptocurrency details.

Code:

```
import requests

cryptocurrency = input("Enter The Crypto Currency Name : ").lower()

url = f"https://api.coingecko.com/api/v3/coins/{cryptocurrency}"

response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    print(f"Details for {cryptocurrency.upper()}:")
    print("Name:", data['name'])
    print("Symbol:", data['symbol'])
    print("Current Price (₹):", data['market_data']
          ['current_price']['inr'])

else:
    print(f"Failed to fetch data for {cryptocurrency}. Status
          code:", response.status_code)
```

Output: (screenshot)

```
/usr/bin/python3 /Users/sakshikore/Desktop/python_lab/7.2cryptocurrency_api.py
sakshikore@Sakshis-MacBook-Air python_lab % /usr/bin/python3 /Users/sakshikore/Desktop/python_lab/7.2cryptocurrency_api.py
/Users/sakshikore/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:34: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
    warnings.warn(
Enter The Crypto Currency Name :
```

Test Case: Any two (screenshot)

```
WARNING:WARN
Enter The Crypto Currency Name : Bitcoin
Details for BITCOIN:
Name: Bitcoin
Symbol: btc
Current Price (₹): 3557611
```

```
Enter The Crypto Currency Name : Ethereum
Details for ETHEREUM:
Name: Ethereum
Symbol: eth
Current Price (₹): 192202
```

Conclusion:

The program prompts the user to enter the name of a cryptocurrency, fetches real-time data from the CoinGecko API, and prints details such as the cryptocurrency's name, symbol, and current price in INR. The script provides a simple yet practical example of interacting with a cryptocurrency API to retrieve relevant information.