INSTITUTE OF TECHNOLOGY AND MANAGEMENT
SKILLS UNIVERSITY,
KHARGHAR, NAVI MUMBAI

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student: Sarthi Sanjaybhai darji

Roll No:12

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

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

	i)
	`*`
	`* *`
	`* * *`
	`* * * *`
	`* * * * *`

	ii) 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5

	iii)
	        *
	      * * *
	    * * * * *
	  * * * * * * *
	* * * * * * * * *

2	2.1 Write a program that define the list of defines the list of define countries that are in BRICS.
	2.2 Write a program to traverse a list in reverse order. 1.By using Reverse method. 2.By using slicing
	2.3 Write a program that scans the email address and forms a tuple of username and domain.
	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]
	2.5 Write a program to compare two dictionaries in Python? (By using == operator)
	2.6 Write a program that creates dictionary of cube of odd numbers in the range.

	2.7 Write a program for various list slicing operation. a= [10,20,30,40,50,60,70,80,90,100] i. Print Complete list ii. Print 4th element of list iii. Print list from0th 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	3.1 Write a program to extend a list in python by using given approach. i. By using + operator. ii. By using Append () iii. By using extend ()
	3.2 Write a program to add two matrices.
	3.3 Write a Python function that takes a list and returns a new list with distinct elements from the first list.
	3.4 Write a program to Check whether a number is perfect or not.
	3.5 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'
4	4.1 Write a program to Create Employee Class & add methods to get employee details & print.

	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 ofeach 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.
	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.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.1

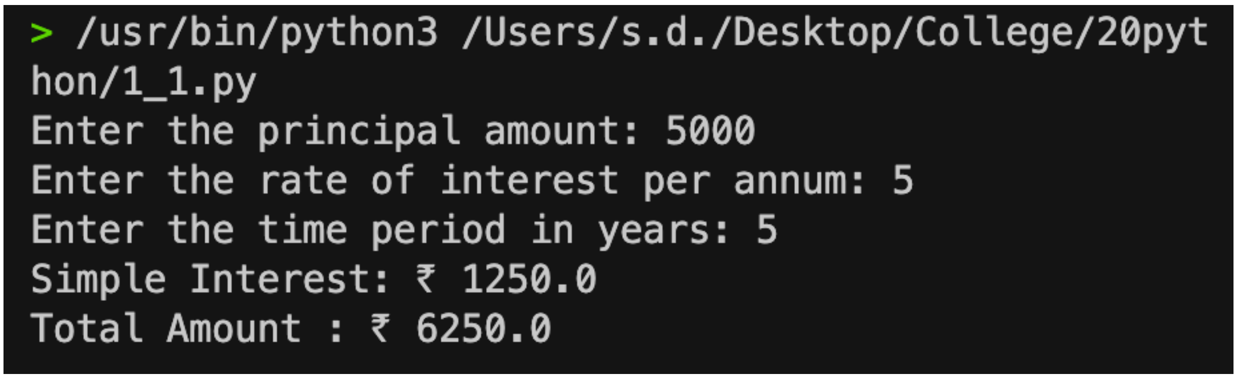
Title: Write a program to compute Simple Interest.

Theory: It's a calculator for simple interest (the basic type of interest). It asks for principal, rate, and time. It multiplies those numbers and divides by 100 to get the interest. It shows you the result in rupees.

Code:

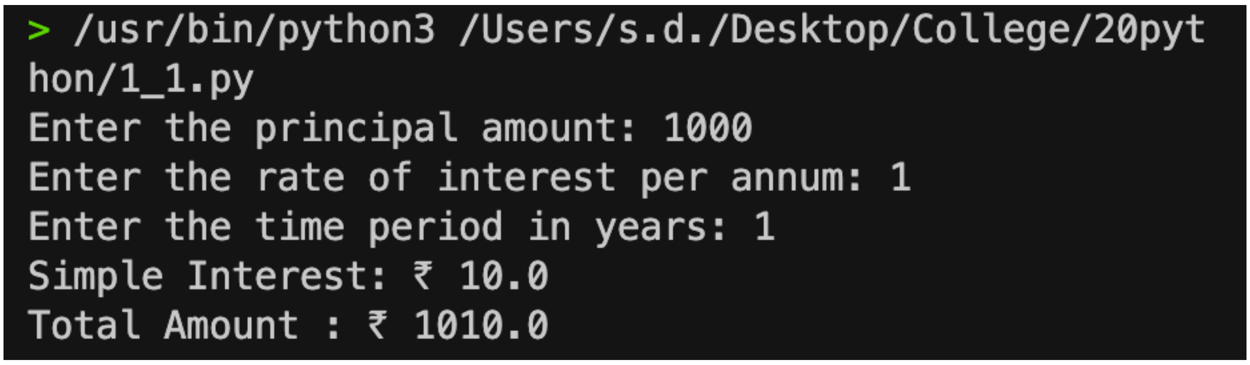
```
principal = float(input("Enter the principal amount: “))

rate = float(input("Enter the rate of interest per annum: “))

time = float(input("Enter the time period in years: “))

simple_interest = (principal * rate * time) / 100

print("Simple Interest: ₹", simple_interest)

print("Total Amount : ₹", simple_interest+principal)
```

Output: (screenshot)

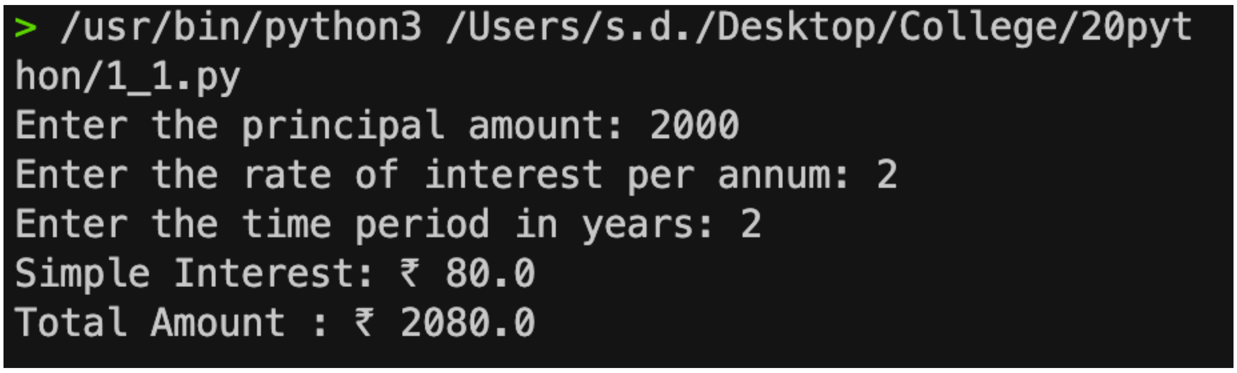


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_1.py
Enter the principal amount: 5000
Enter the rate of interest per annum: 5
Enter the time period in years: 5
Simple Interest: ₹ 1250.0
Total Amount : ₹ 6250.0
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_1.py
Enter the principal amount: 1000
Enter the rate of interest per annum: 1
Enter the time period in years: 1
Simple Interest: ₹ 10.0
Total Amount : ₹ 1010.0
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_1.py
Enter the principal amount: 2000
Enter the rate of interest per annum: 2
Enter the time period in years: 2
Simple Interest: ₹ 80.0
Total Amount : ₹ 2080.0
```

Conclusion: This code effectively calculates simple interest based on user-provided values. It demonstrates basic input, calculation, and output operations in Python. It's a simple but useful tool for financial calculations.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.2

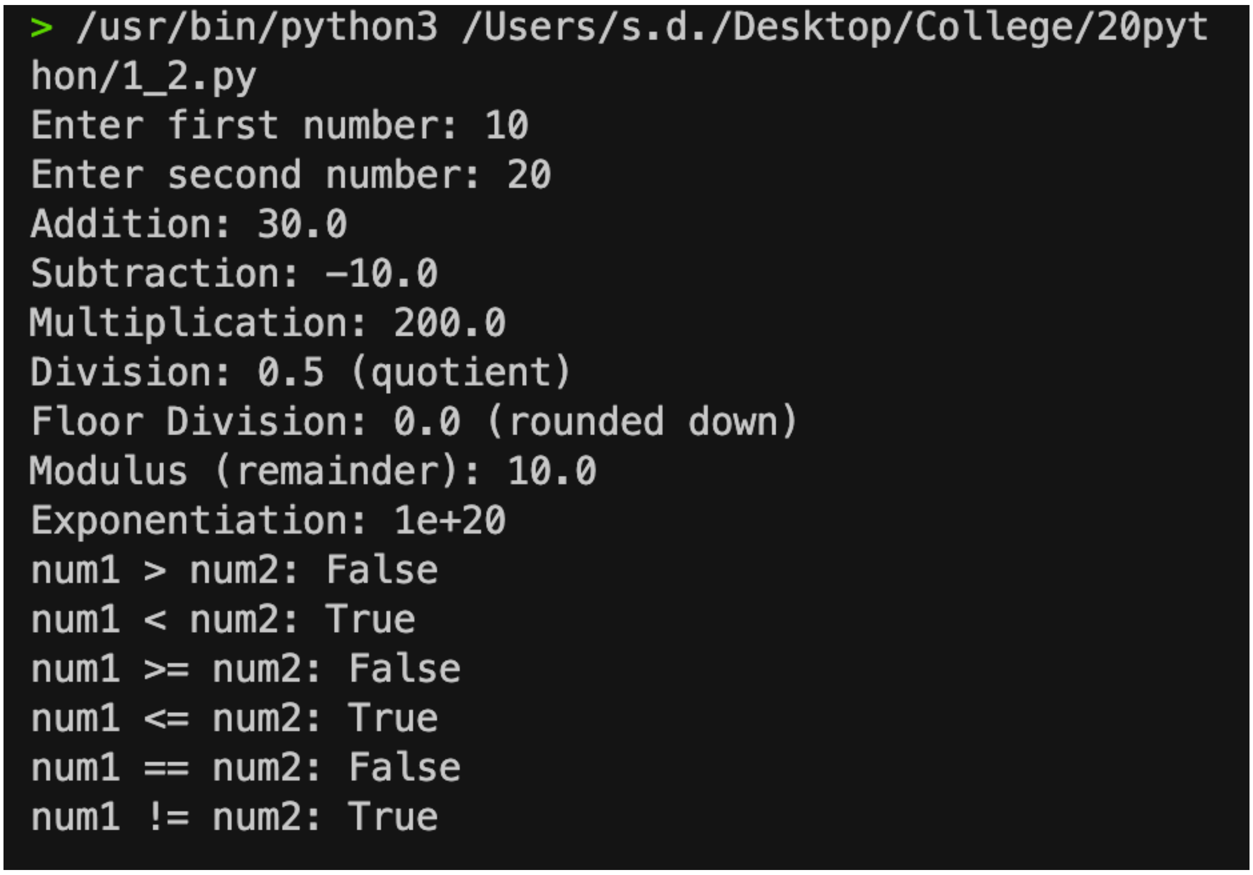
Title: Write a program to perform arithmetic, Relational operators.

Theory: It's a calculator and a number comparer in one. It does basic math like adding, subtracting, multiplying, and dividing. It also tells you if one number is bigger, smaller, or equal to another.

Code:

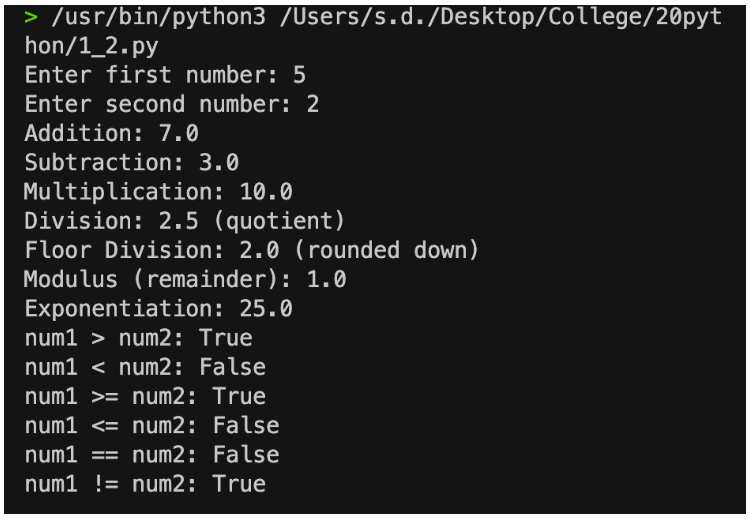
```
# Arithmetic operators

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))



print("Addition:", num1 + num2)

print("Subtraction:", num1 - num2)

print("Multiplication:", num1 * num2)

print("Division:", num1 / num2, "(quotient)")

print("Floor Division:", num1 // num2, "(rounded down)")

print("Modulus (remainder):", num1 % num2)

print("Exponentiation:", num1 ** num2)



# Relational operators

print("num1 > num2:", num1 > num2)

print("num1 < num2:", num1 < num2)
print("num1 >= num2:", num1 >= num2)

print("num1 <= num2:", num1 <= num2)

print("num1 == num2:", num1 == num2)  # Equal to

print("num1 != num2:", num1 != num2)  # Not equal to
```

Output: (screenshot)

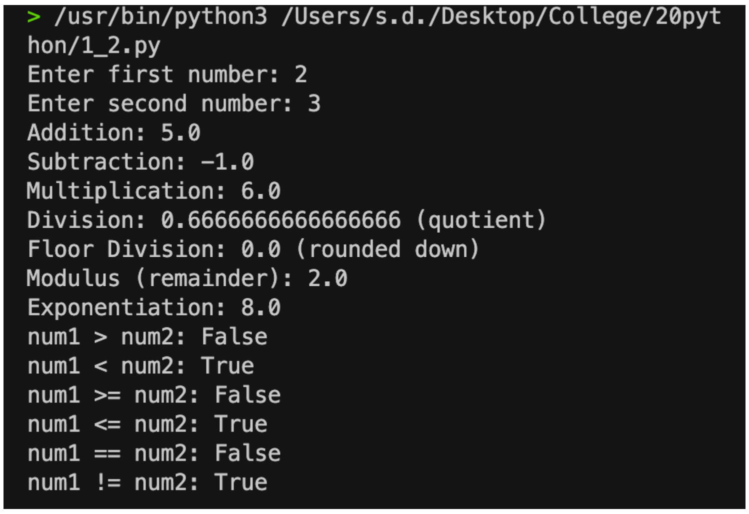


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_2.py
Enter first number: 10
Enter second number: 20
Addition: 30.0
Subtraction: -10.0
Multiplication: 200.0
Division: 0.5 (quotient)
Floor Division: 0.0 (rounded down)
Modulus (remainder): 10.0
Exponentiation: 1e+20
num1 > num2: False
num1 < num2: True
num1 >= num2: False
num1 <= num2: True
num1 == num2: False
num1 != num2: True
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_2.py
Enter first number: 5
Enter second number: 2
Addition: 7.0
Subtraction: 3.0
Multiplication: 10.0
Division: 2.5 (quotient)
Floor Division: 2.0 (rounded down)
Modulus (remainder): 1.0
Exponentiation: 25.0
num1 > num2: True
num1 < num2: False
num1 >= num2: True
num1 <= num2: False
num1 == num2: False
num1 != num2: True
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_2.py
Enter first number: 2
Enter second number: 3
Addition: 5.0
Subtraction: -1.0
Multiplication: 6.0
Division: 0.6666666666666666 (quotient)
Floor Division: 0.0 (rounded down)
Modulus (remainder): 2.0
Exponentiation: 8.0
num1 > num2: False
num1 < num2: True
num1 >= num2: False
num1 <= num2: True
num1 == num2: False
num1 != num2: True
```

Conclusion : This code demonstrates various arithmetic and relational operators in Python. It showcases how to perform numerical calculations and comparisons. It's a valuable tool for understanding fundamental programming concepts and techniques.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.3

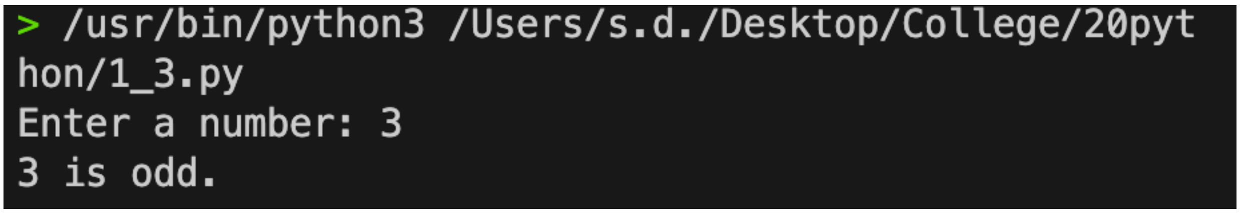
Title:Write a program to find whether a given no is even & odd.

Theory: It's a number detective that checks for even or odd numbers. It uses a special trick (the modulo operator) to see if a number divides evenly by 2. It prints a clear message to tell you the result.

Code:

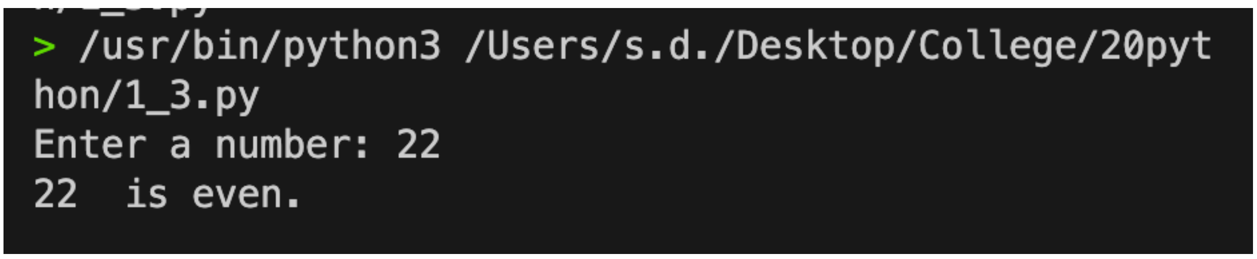
```
number = int(input("Enter a number: "))

if number % 2 == 0:

    print(number," is even.")

else:

    print(number,"is odd.”)
```

Output: (screenshot)

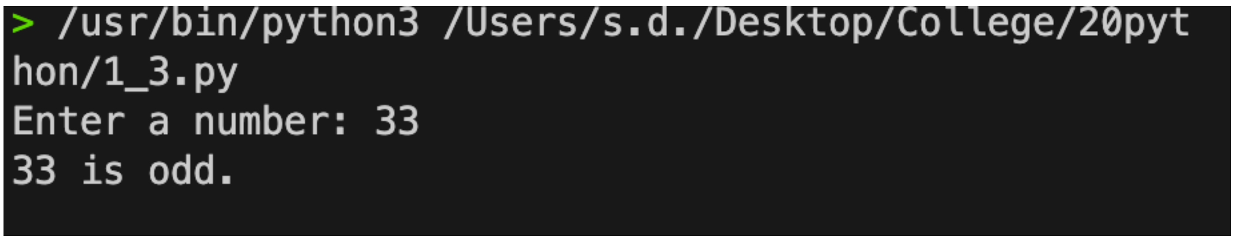


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_3.py
Enter a number: 3
3 is odd.
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_3.py
Enter a number: 22
22  is even.
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_3.py
Enter a number: 33
33 is odd.
```

Conclusion : It's a number detective that checks for even or odd numbers. It uses a special trick (the modulo operator) to see if a number divides evenly by 2. It prints a clear message to tell you the result. It's a building block for more complex math and programming ideas.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.4

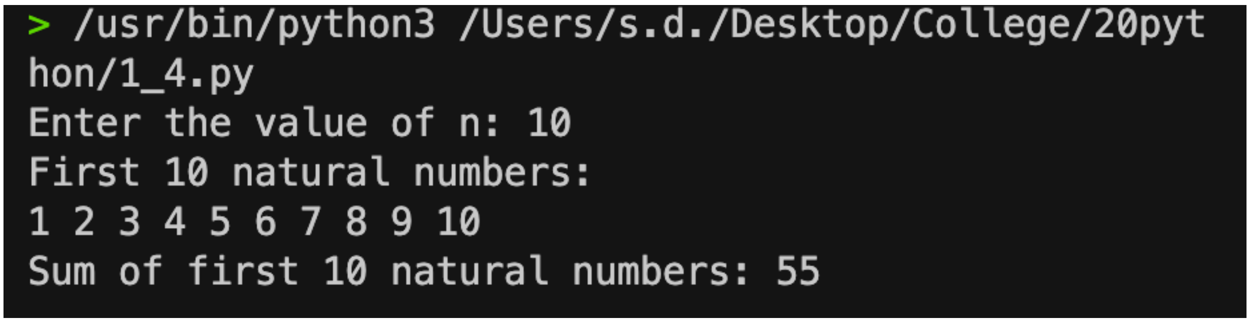
Title: Write a program to print first n natural number & their sum.

Theory: It's a counter and adder that works with natural numbers (1, 2, 3, ...). You tell it how many numbers to count (n), and it does two things. Prints each number from 1 to n, one by one. Adds up those numbers and tells you the total sum. It uses a loop to repeat the counting and adding process.

Code:

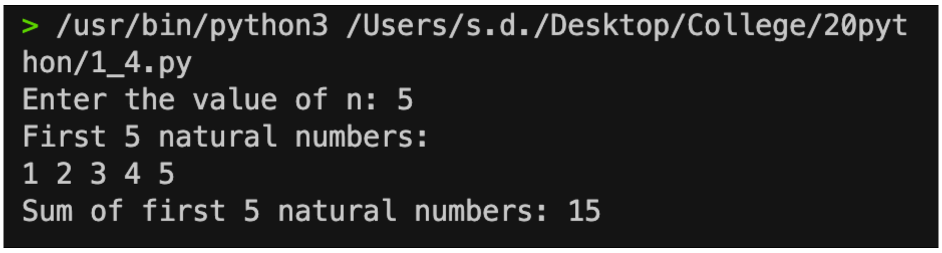
```
n = int(input("Enter the value of n: "))

sum = 0

print("First", n, "natural numbers:")

for i in range(1, n + 1):

   print( i, end=“ ")

   sum =sum+i

print("\nSum of first", n, "natural numbers:", sum)
```

Output: (screenshot)

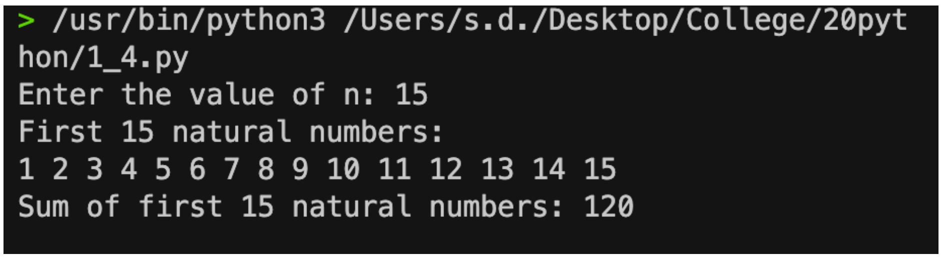


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_4.py
Enter the value of n: 10
First 10 natural numbers:
1 2 3 4 5 6 7 8 9 10
Sum of first 10 natural numbers: 55
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_4.py
Enter the value of n: 5
First 5 natural numbers:
1 2 3 4 5
Sum of first 5 natural numbers: 15
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_4.py
Enter the value of n: 15
First 15 natural numbers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Sum of first 15 natural numbers: 120
```

Conclusion : This code effectively prints the first n natural numbers and calculates their sum. It demonstrates the use of for loops for iteration and the range function for generating sequences. It showcases variable initialization and accumulation for calculating results. It's a basic but essential concept in programming and numerical calculations.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.5

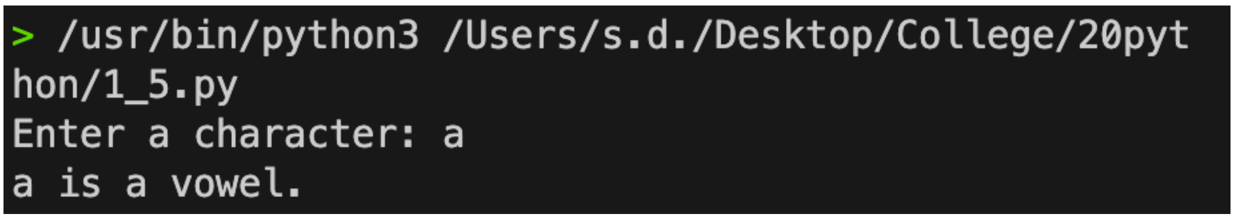
Title: Write a program to determine whether the character entered is a Vowel or not.

Theory: It's a vowel finder that checks if a letter is a vowel (a, e, i, o, u). It has a list of all vowels to compare against. It uses a special trick (the in operator) to see if the letter is in the list. It prints a clear message to tell you the result.

Code :

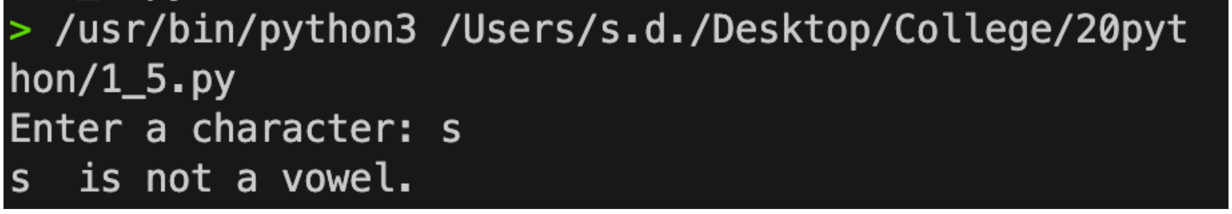
```
vowels = "aeiouAEIOU"

character = input("Enter a character: ")

if character in vowels:

    print(character,"is a vowel.")

else:

    print(character," is not a vowel.”)
```

Output: (screenshot)

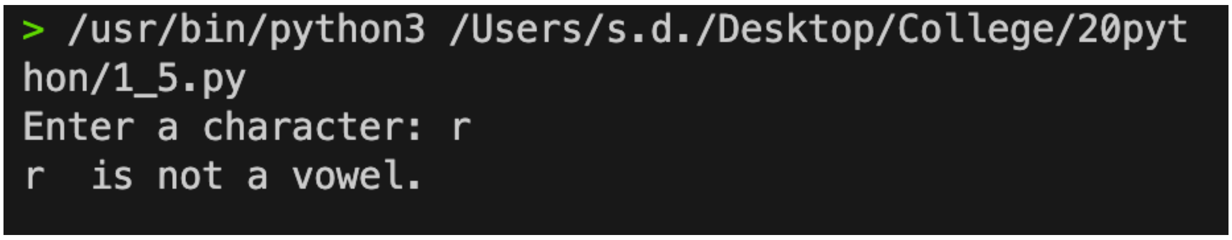


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_5.py
Enter a character: a
a is a vowel.
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_5.py
Enter a character: s
s  is not a vowel.
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_5.py
Enter a character: r
r  is not a vowel.
```

Conclusion : This code effectively determines whether a given character is a vowel. It demonstrates string membership testing using the in operator. It showcases conditional statements (if-else) for decision-making. It's a simple but useful tool for character classification and text analysis.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.6

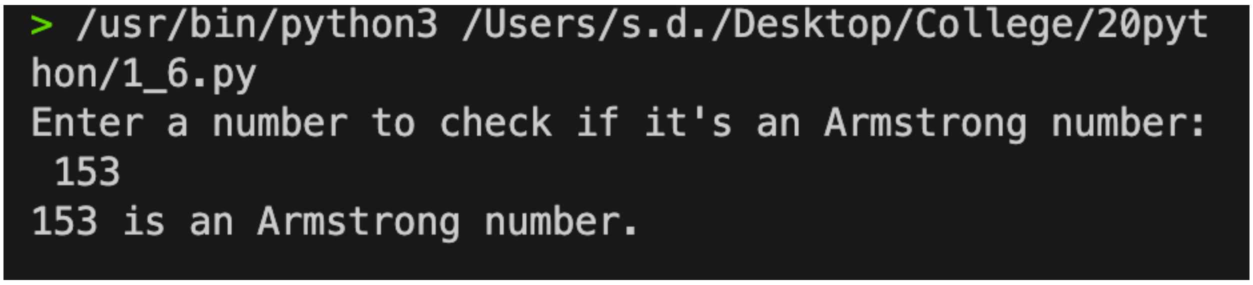
Title: Write a program to find whether given number is an Armstrong Number.

Theory: It's a number puzzle solver that checks for special numbers called Armstrong numbers. It has a secret formula to test if a number is Armstrong or not. It asks you for a number to check. It does some math magic to see if the number matches the formula. It tells you if the number is Armstrong or not.

Code :

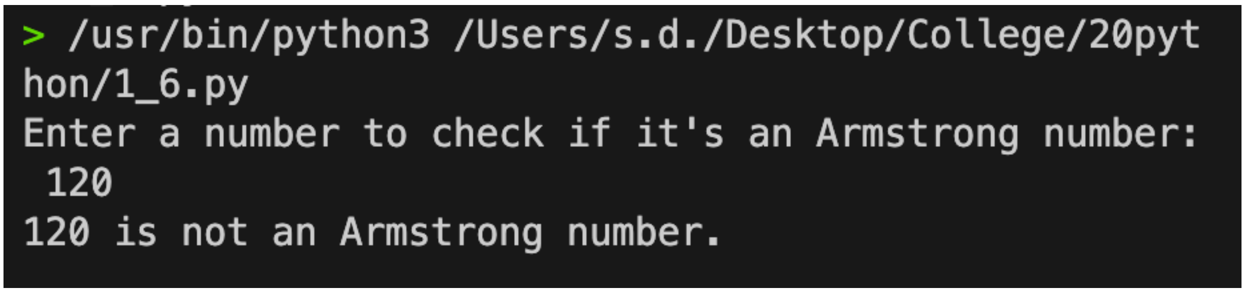
```
def is_armstrong(n):

    original_num = n

    sum_of_digits = 0

    while n > 0:

        digit = n % 10

        sum_of_digits =sum_of_digits+ digit**len(str(original_num))

        n //= 10

    return sum_of_digits == original_num

number = int(input("Enter a number to check if it's an Armstrong number: "))

if is_armstrong(number):

  print(number ,"is an Armstrong number.")

else:

  print(number ,"is not an Armstrong number.”)
```

Output: (screenshot)

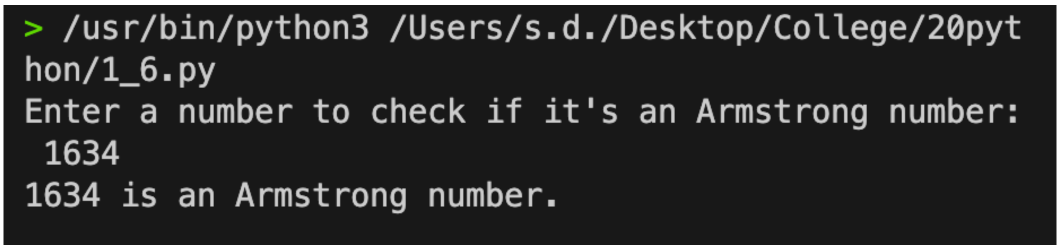


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_6.py
Enter a number to check if it's an Armstrong number:
 153
153 is an Armstrong number.
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_6.py
Enter a number to check if it's an Armstrong number:
 120
120 is not an Armstrong number.
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_6.py
Enter a number to check if it's an Armstrong number:
 1634
1634 is an Armstrong number.
```

Conclusion : This code effectively determines whether a given number is an Armstrong number. It demonstrates function definition, loops, modulo operation, exponentiation, and conditional statements. It incorporates user input to make it interactive.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.7

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

Theory: Factorial: It's a mathematical operation that involves multiplying a number by all positive integers smaller than it. For example, 5! (5 factorial) is 5 * 4 * 3 * 2 * 1 = 120. The code calculates the factorial using a for loop, which means it repeats the multiplication process step by step until it reaches the desired number.

Code :

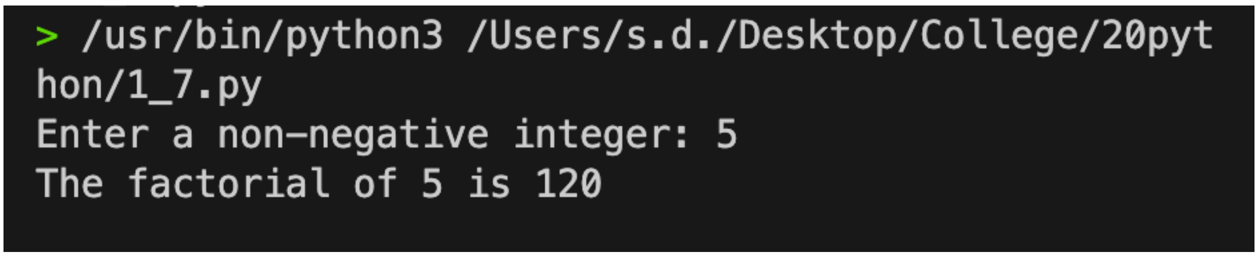
```
def factorial(n):

    if n < 0:

        return "Factorial is not defined for negative numbers."

    elif n == 0:

        return 1

    else:

        factorial = 1

        for i in range(1, n + 1):

            factorial *= i

        return factorial
```

number = int(input("Enter a non-negative integer: "))

result = factorial(number)

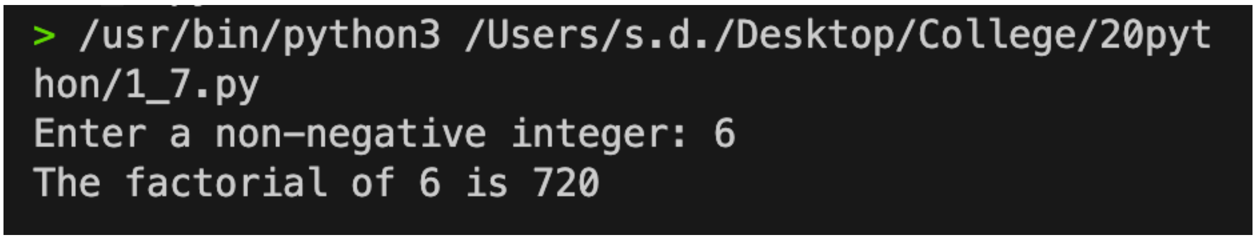
print("The factorial of", number, "is", result)

Output: (screenshot)

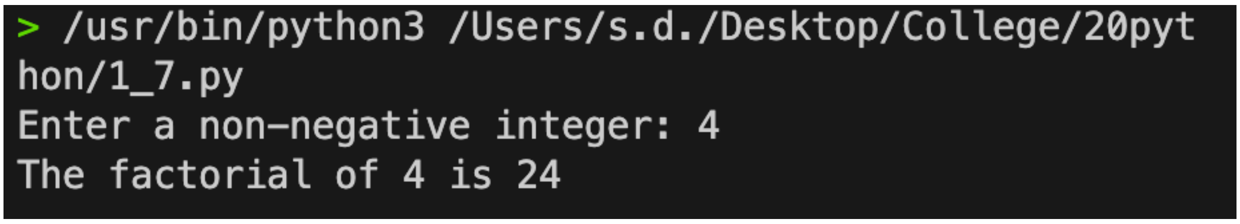


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_7.py
Enter a non-negative integer: 5
The factorial of 5 is 120
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_7.py
Enter a non-negative integer: 6
The factorial of 6 is 720
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_7.py
Enter a non-negative integer: 4
The factorial of 4 is 24
```

Conclusion : The provided code effectively calculates the factorial of a non-negative integer using a for loop, demonstrating a fundamental programming concept. It handles negative input gracefully and returns an informative message. The code is well-structured, readable, and includes clear comments for explanation.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.8.1

Title: Write a program to print the following pattern

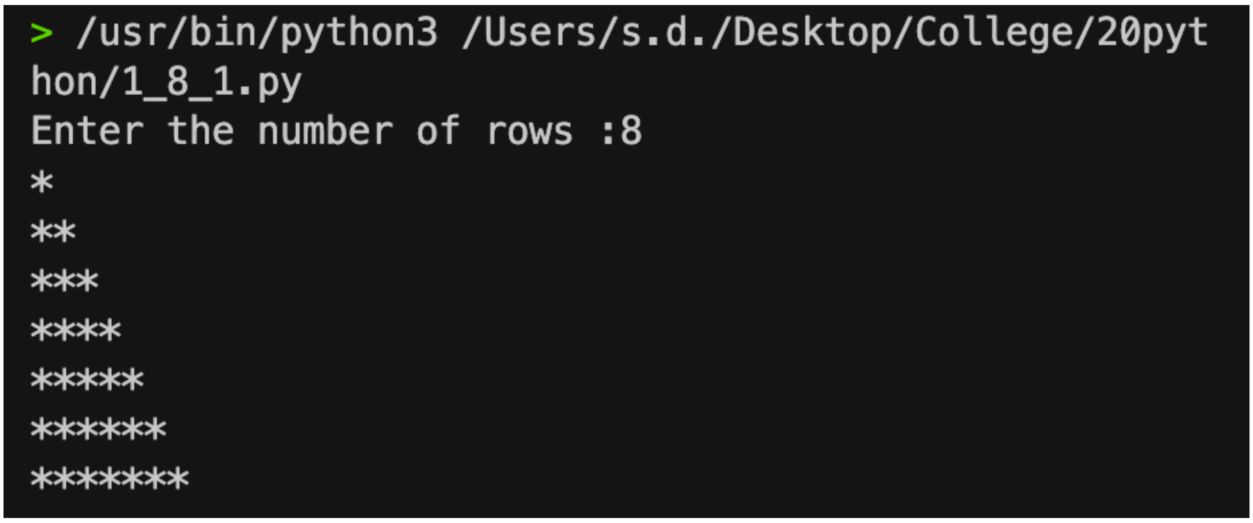
```
*

* *

* * *

* * * *

* * * * *
```

Theory: We use a loop to count from 1 to 5. For each number, we print that number of stars using string multiplication. This creates a pyramid pattern with an increasing number of stars in each row.

Code :

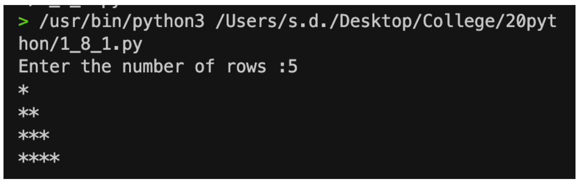
```
n=int(input("Enter the number of rows :"))

for i in range(1, n):

    print( "*" * i )
```

Output: (screenshot)

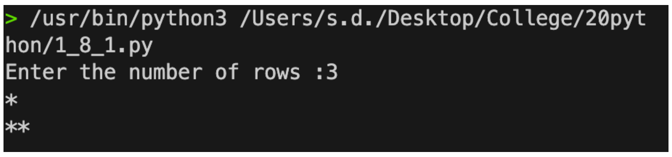


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_1.py
Enter the number of rows :8
*
**
***
****
*****
******
*******
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_1.py
Enter the number of rows :5
*
**
***
****
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_1.py
Enter the number of rows :3
*
**
```

Conclusion: The code effectively demonstrates how to create a simple pattern using a for loop and string multiplication. It's concise and readable, with clear indentation for understanding the loop structure. It could be made more flexible by allowing the user to specify the number of rows or the character used to create the pattern.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.8.2

Title:Write a program to print the following pattern

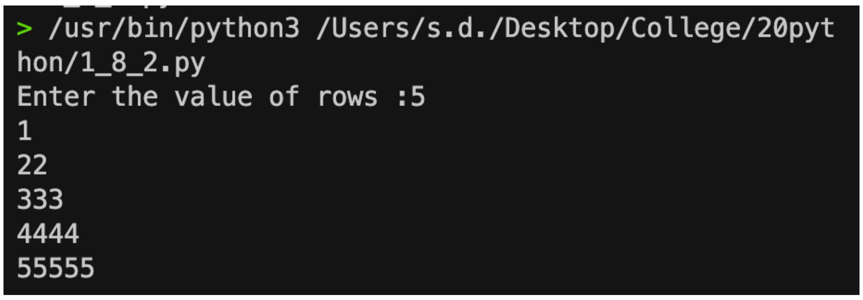
```
 1

 2 2

 3 3 3

4 4 4 4

5 5 5 5 5
```

Theory: For loops and range(): Control the number of iterations using a for loop and range() function.Create spaces using string multiplications for proper alignment.Convert numbers to strings to enable string operations.Repeat numbers using string multiplication to create the pattern.

Code :

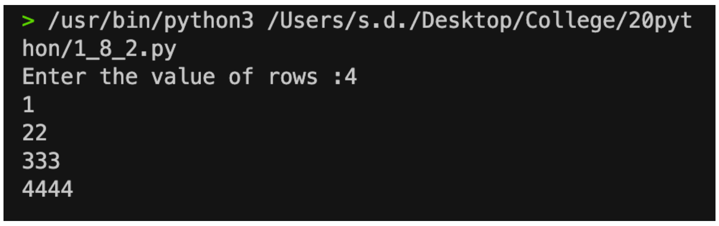
```
n=int(input("Enter the value of rows :"))

for i in range(1, n+1):

    print(str(i) * i+" " * (n+1 - i))
```

Output: (screenshot)

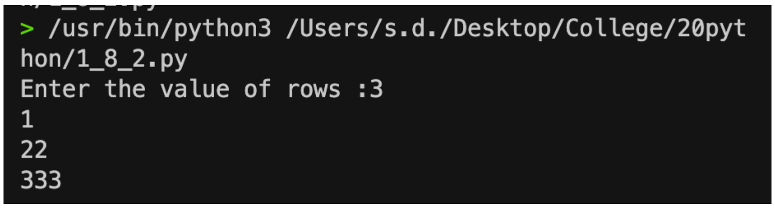


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_2.py
Enter the value of rows :5
1
22
333
4444
55555
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_2.py
Enter the value of rows :4
1
22
333
4444
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_2.py
Enter the value of rows :3
1
22
333
```

Conclusion: The code effectively generates the desired pattern using string manipulation within a for loop. It demonstrates combining string operations and loops for pattern creation. It could be made more flexible by allowing user input for the number of rows or the character used for the pattern.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1.8.3

Title:Write a program to print the following pattern

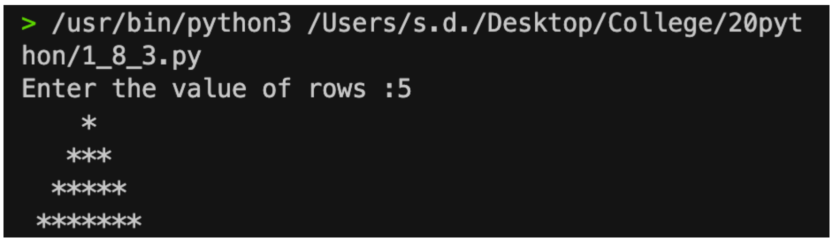
```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
```

Theory: Create spaces using string multiplication (" " * (n - i)) for alignment. Create stars using string multiplication ("*" * (2 * i - 1)) to form the pattern. Combine spaces and stars to construct each row.

Code :

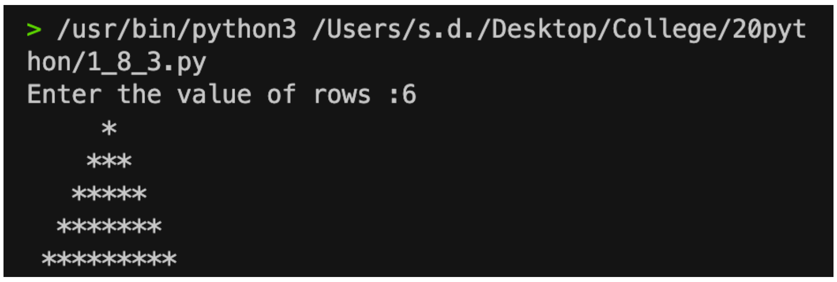
```
n=int(input("Enter the value of rows :"))

for i in range(1, n):

    print(" " * (n - i) + "*" * (2 * i - 1))
```

Output: (screenshot)

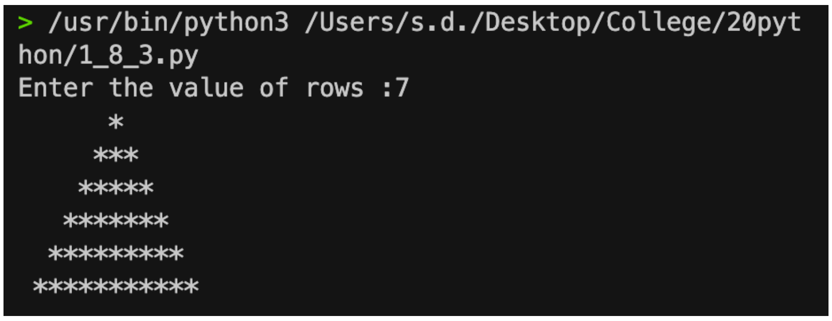


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_3.py
Enter the value of rows :5
    *
   ***
  *****
 *******
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_3.py
Enter the value of rows :6
     *
    ***
   *****
  *******
 *********
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/1_8_3.py
Enter the value of rows :7
      *
     ***
    *****
   *******
  *********
 ***********
```

Conclusion : The code successfully generates a symmetrical pyramid pattern with spaces and stars. It demonstrates effective use of string manipulation and loops for pattern creation. It could be made more flexible by allowing user input for the number of rows or characters used. This technique can be applied to create various patterns of different shapes and sizes**.**

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2.1

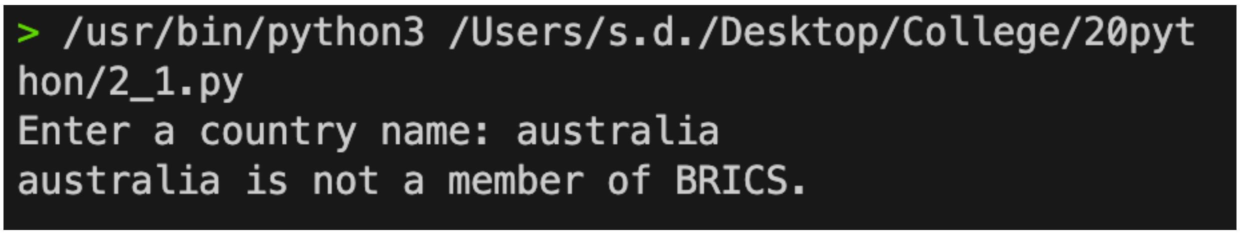
Title: Write a program that define the list of defines the list of define countries that are in BRICS.

Theory: It's a country checker that knows about the BRICS countries. It has a list of BRICS countries stored in its memory. It asks you for a country name to check. It looks through the list to see if the country is there. It tells you if the country is a BRICS member or not.

Code :

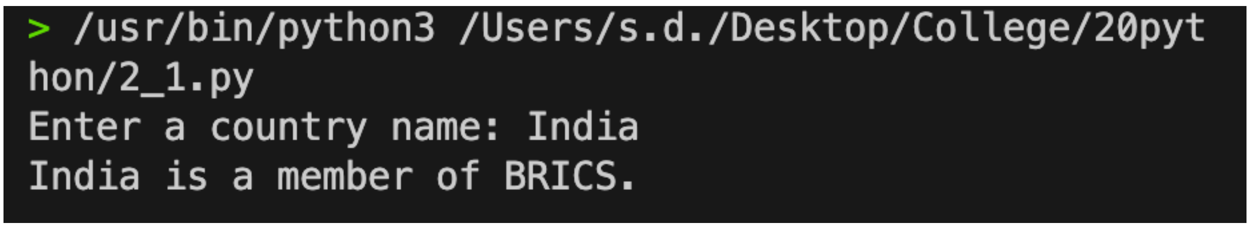
```
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]

country_name = str(input("Enter a country name: "))

if country_name in brics_countries:

    print(country_name, "is a member of BRICS.")

else:

    print(country_name, "is not a member of BRICS.")
```

Output: (screenshot)

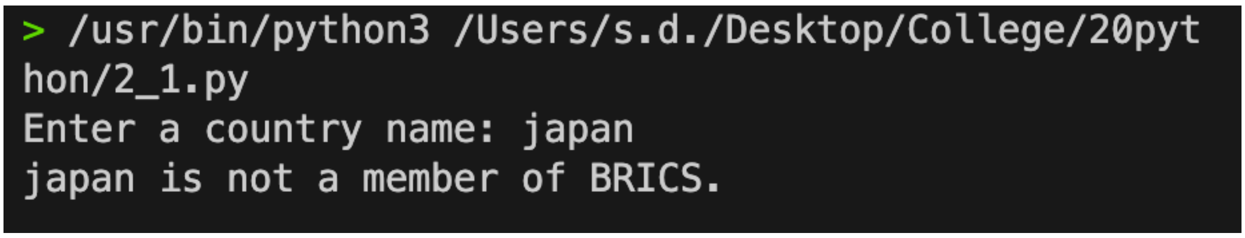


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_1.py
Enter a country name: australia
australia is not a member of BRICS.
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_1.py
Enter a country name: India
India is a member of BRICS.
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_1.py
Enter a country name: japan
japan is not a member of BRICS.
```

Conclusion: This code effectively determines whether a given country is a BRICS member. It demonstrates list creation, membership testing, user input, and conditional statements. It's a simple but useful tool for checking country

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2.2

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

1.By using Reverse method.

2.By using slicing

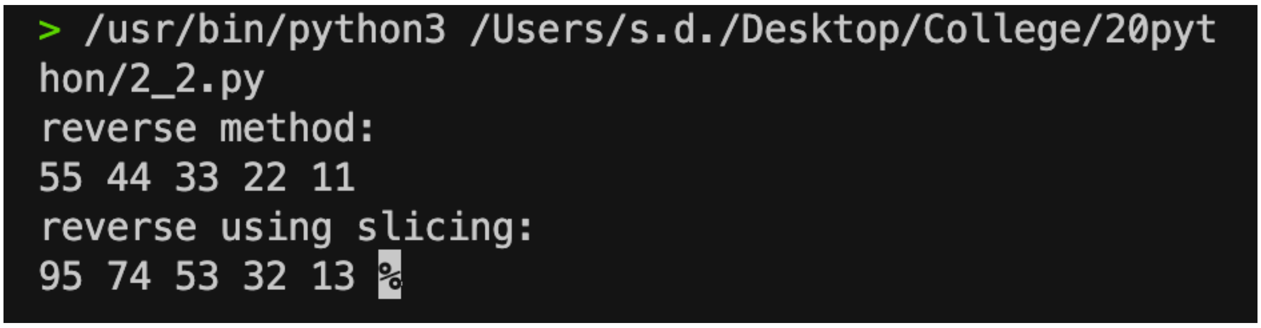
Theory: Accessing elements in the opposite order they are stored. Rearrange elements in a list in-place, directly modifying the original list. Extracting specific portions of a list using a range-like syntax, creating new views without modifying the original list. A negative step value iterates in reverse order.

Code :

```
#reverse method

my_list1 = [11, 22, 33, 44, 55]

my_list1.reverse()



print("reverse method:")

for item in my_list1:

    print(item ,end=" ")

print()



#slice method

my_list2 = [13, 32,53,74, 95]
```

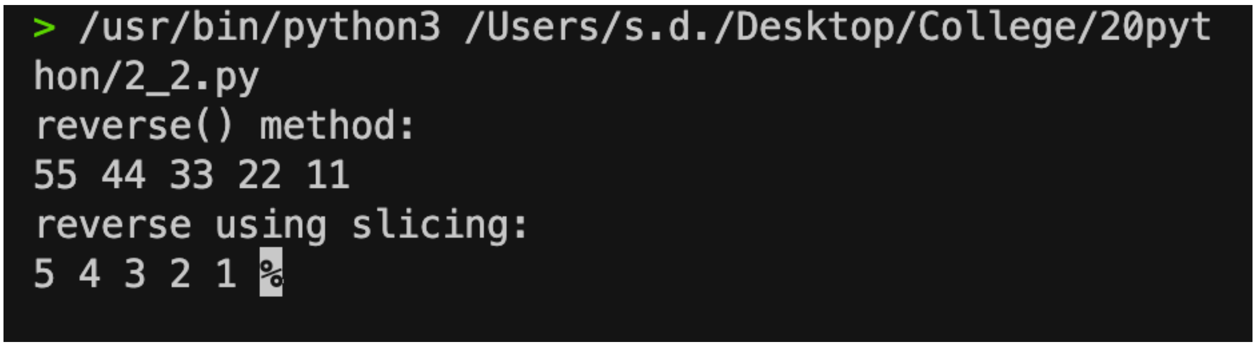
```
print("reverse using slicing:")

for item in my_list2[::-1]:

    print(item, end=" “)
```

Output: (screenshot)

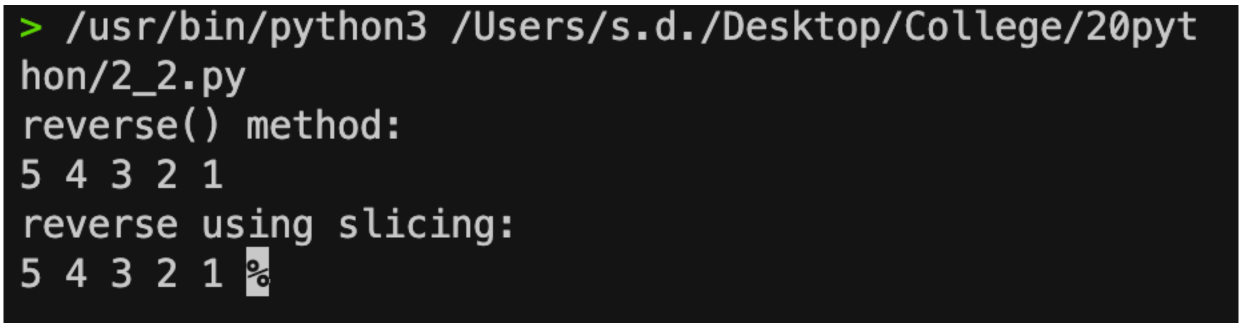


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_2.py
reverse method:
55 44 33 22 11
reverse using slicing:
95 74 53 32 13 %
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_2.py
reverse() method:
55 44 33 22 11
reverse using slicing:
5 4 3 2 1 %
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_2.py
reverse() method:
5 4 3 2 1
reverse using slicing:
5 4 3 2 1 %
```

Conclusion : Efficient for in-place reversal, but modifies the original list. Preserves the original list, creates a temporary reversed view. The choice depends on whether you need to modify the original list or maintain its integrity. Both methods effectively achieve reverse traversal, producing identical output.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2.3

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

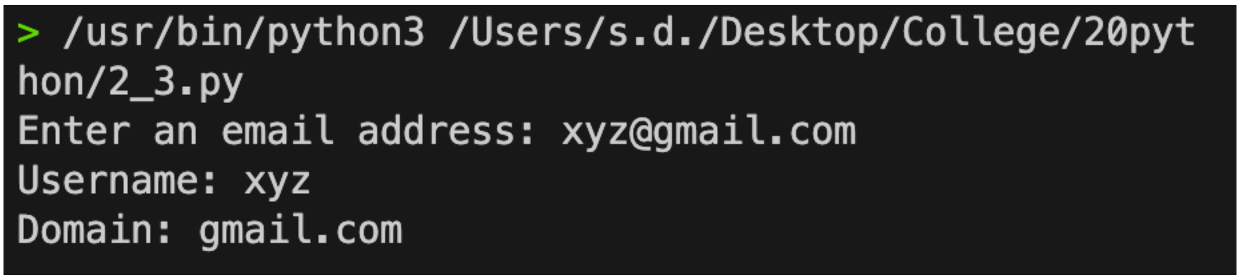
Theory: Code finds username and domain in email. Uses "@" symbol to split email into two parts. Stores username and domain in a special group called a tuple. Checks for valid emails before splitting. If invalid, tells you the email is wrong.

Code :

```
def domainn(email):

  index = email.find("@")

  if index != -1:

    username = email[:index]

    domain = email[index + 1:]

    return username, domain

  else:

    print("Error")



email = input("Enter an email address: ")

username, domain = domainn(email)

if username and domain:

  print("Username:", username)

  print("Domain:", domain)
```

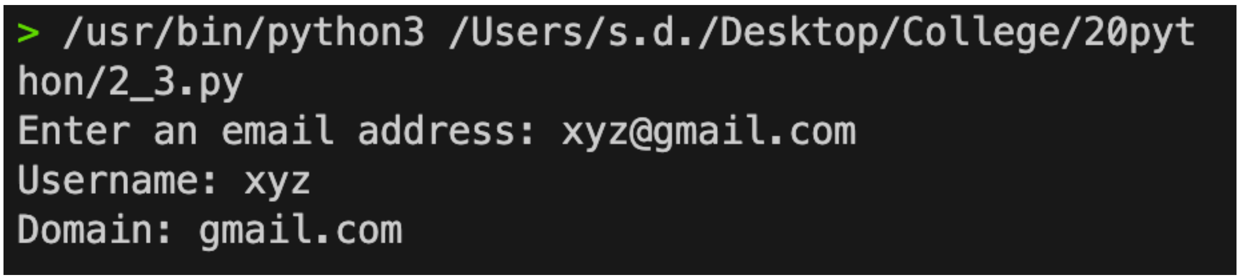
```
else:

  print("Invalid email address.”)
```

Output: (screenshot)

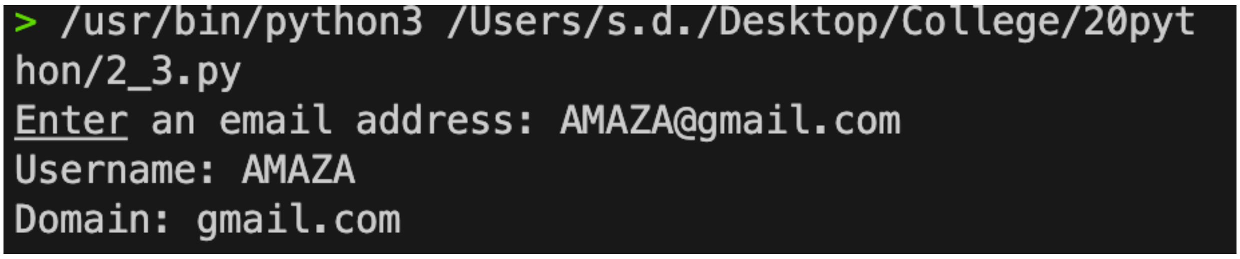


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_3.py
Enter an email address: xyz@gmail.com
Username: xyz
Domain: gmail.com
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_3.py
Enter an email address: xyz@gmail.com
Username: xyz
Domain: gmail.com
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_3.py
Enter an email address: AMAZA@gmail.com
Username: AMAZA
Domain: gmail.com
```

Conclusion : This code effectively extracts the username and domain from a valid email address, demonstrating . String manipulation techniques for finding and extracting substrings. Function definition for modularity and reusability. Tuples for returning multiple values. User input and output for interactivity.

Name of Student: Sarthi Sanjaybhai Darji
Roll Number: 12

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.

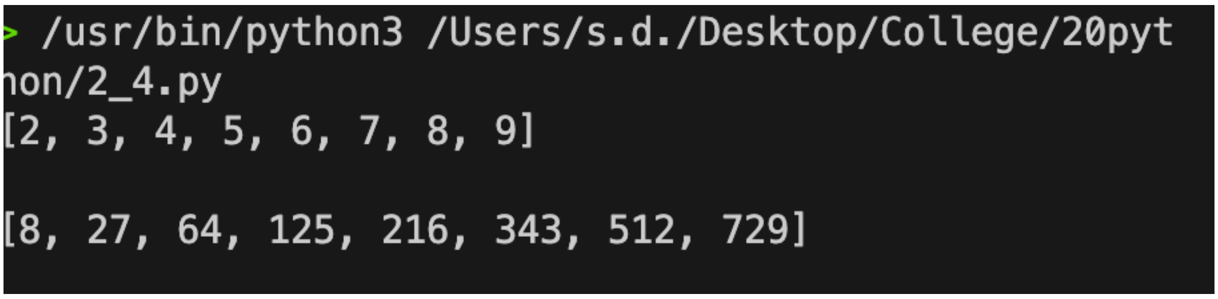
i/p: c= [2,3,4,5,6,7,8,9]

Theory: It takes a list of numbers and creates a new list with special pairs. create lists of cube

Code :

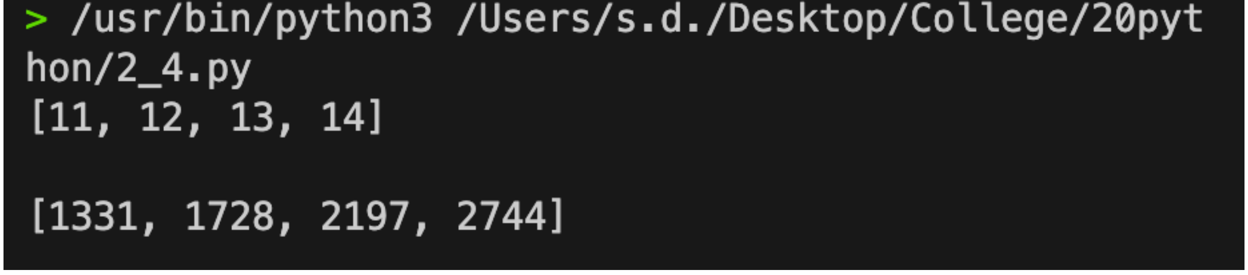
```
c = [2, 3, 4, 5, 6, 7, 8, 9]

cube = [(num**3) for num in c]

print(c)

print()

print(cube)
```

Output: (screenshot)

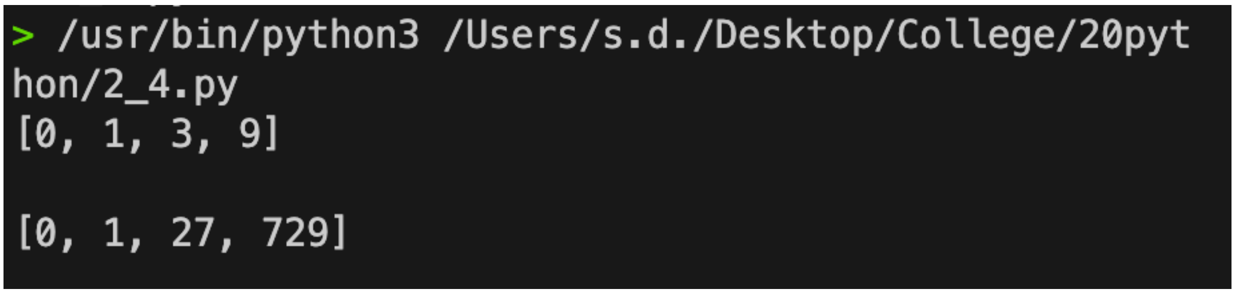


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
non/2_4.py
[2, 3, 4, 5, 6, 7, 8, 9]

[8, 27, 64, 125, 216, 343, 512, 729]
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_4.py
[11, 12, 13, 14]

[1331, 1728, 2197, 2744]
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_4.py
[0, 1, 3, 9]

[0, 1, 27, 729]
```

Conclusion : This code efficiently creates a list of tuples, each containing a number from the input list and its corresponding cube. It demonstrates list comprehension, a powerful tool for concise list creation and transformation. It highlights the use of tuples to group related data, such as a number and its cube.a

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2.5

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

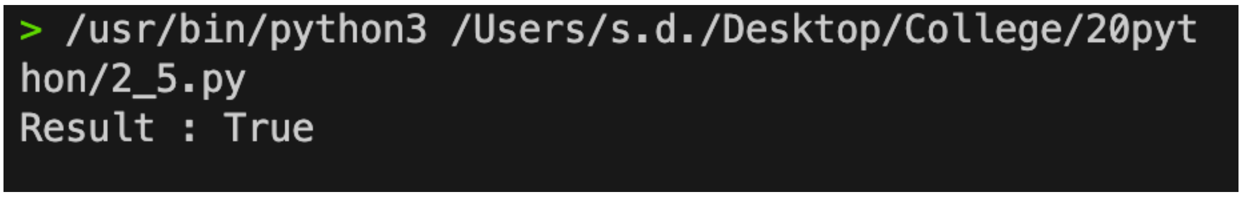
(By using == operator)

Theory : It's a dictionary detective that checks if two dictionaries are twins! .It looks at each key-value pair in both dictionaries to see if they match perfectly.

Code :

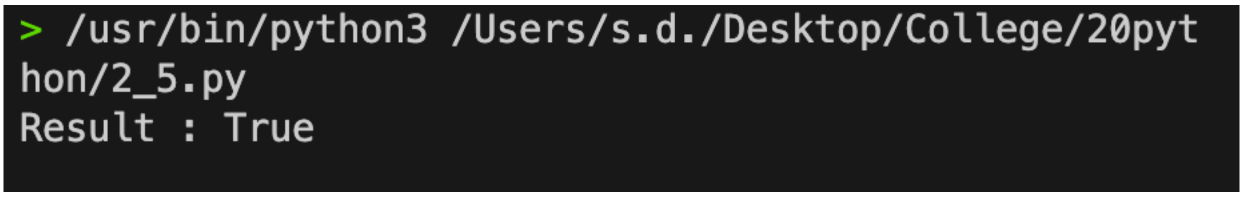
```
dict1 = {"name": "sarthi", "age": 17}

dict2 = {"name": "sarthi", "age": 17}

result = dict1 == dict2

print("Result :",result)
```

Output: (screenshot)

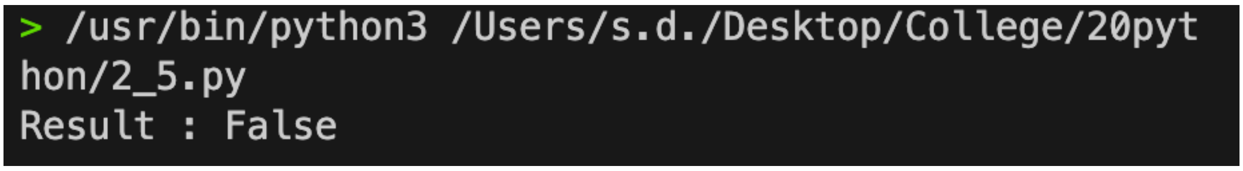


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_5.py
Result : True
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_5.py
Result : True
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_5.py
Result : False
```

Conclusion : This code effectively compares two dictionaries using the == operator for equality.Same keys and values in the same order.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2.6

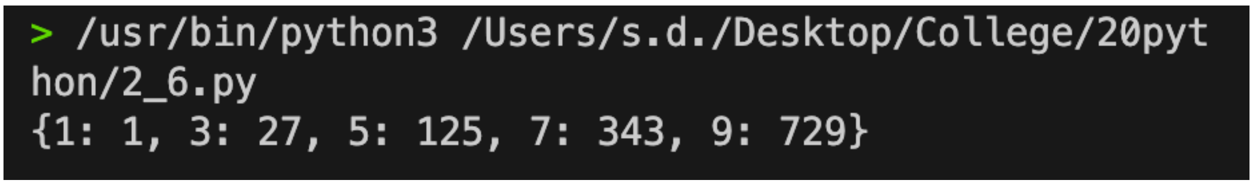
Title: Write a program that creates dictionary of cube of odd numbers in the range.

Theory : It's a cube maker and number organizer. It builds a special dictionary with numbers as keys and their cubes (numbers multiplied by themselves three times) as values. It only puts odd numbers in the dictionary, like a club for odd numbers only. It uses a loop to go through a range of numbers, like counting on a number line.

Code :

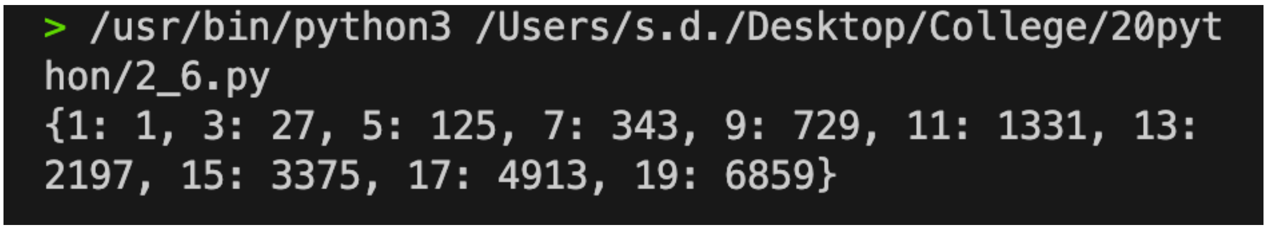
```
def cube_range( a , b):

    cubes= {}

    for num in range(a , b+1):

        if num%2!=0:
            cubes[num] = num**3

    return cubes

cubes = cube_range(1,5)

print(cubes)
```

Output: (screenshot)

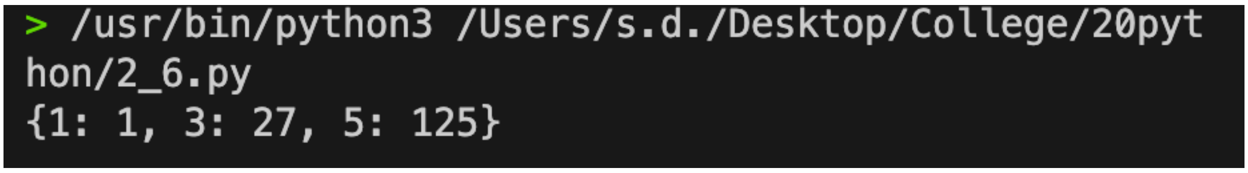


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_6.py
{1: 1, 3: 27, 5: 125, 7: 343, 9: 729}
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_6.py
{1: 1, 3: 27, 5: 125, 7: 343, 9: 729, 11: 1331, 13:
2197, 15: 3375, 17: 4913, 19: 6859}
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/2_6.py
{1: 1, 3: 27, 5: 125}
```

Conclusion : This code efficiently creates a dictionary of cubes of odd numbers within a specified range. Dictionary creation and manipulation. Conditional statements for filtering odd numbers.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2.7

Title: Write a program for various list slicing operations

a= [10,20,30,40,50,60,70,80,90,100]

i. Print Complete list

ii. Print 4th element of list

iii. Print list from0th 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.

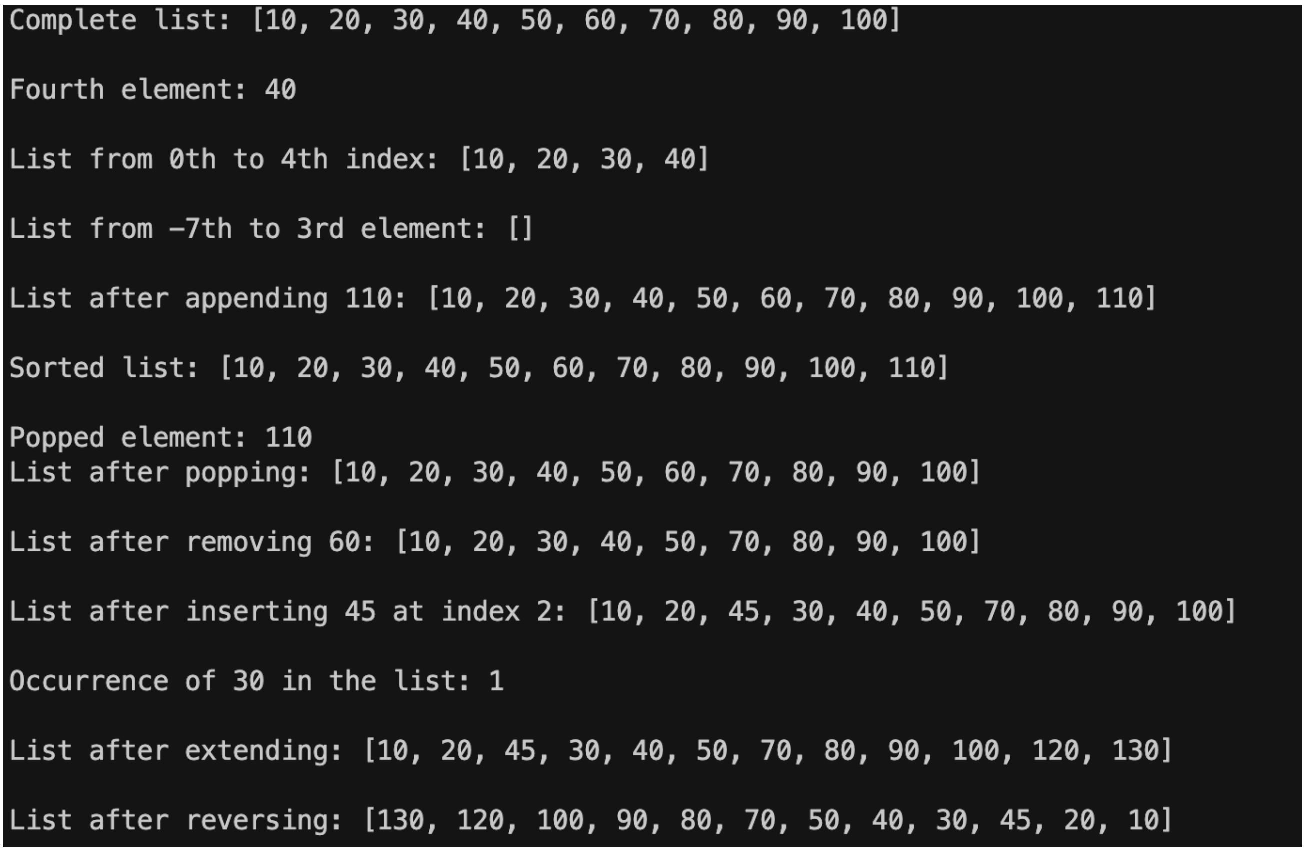
Code:

```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

print("Complete list:", a)

print()

print("Fourth element:", a[3])

print()

print("List from 0th to 4th index:", a[:4])

print()

print("List from -7th to 3rd element:", a[-7:3])

print()

a.append(110)

print("List after appending 110:", a)

print()

a.sort()

print("Sorted list:", a)



print()

element = a.pop()

print("Popped element:", element)
print("List after popping:", a)

print()



a.remove(60)

print("List after removing 60:", a)

print()



a.insert(2, 45)

print("List after inserting 45 at index 2:", a)



print()

count = a.count(30)

print("Occurrence of 30 in the list:", count)



print()

a.extend([120, 130])

print("List after extending:", a)



print()
```

a.reverse()

print("List after reversing:", a)

Output: (screenshot)



```
Complete list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Fourth element: 40

List from 0th to 4th index: [10, 20, 30, 40]

List from -7th to 3rd element: []

List after appending 110: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]

Sorted list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]

Popped element: 110
List after popping: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

List after removing 60: [10, 20, 30, 40, 50, 70, 80, 90, 100]

List after inserting 45 at index 2: [10, 20, 45, 30, 40, 50, 70, 80, 90, 100]

Occurrence of 30 in the list: 1

List after extending: [10, 20, 45, 30, 40, 50, 70, 80, 90, 100, 120, 130]

List after reversing: [130, 120, 100, 90, 80, 70, 50, 40, 30, 45, 20, 10]
```

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 3.1

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

i. By using + operator.

ii. By using Append ()

iii. By using extend ()

Theory : It's a list stretcher! It makes lists longer using different tricks. The + sign is like gluing two lists together, but it makes a whole new list. append() is like adding a single block to a tower, one at a time. extend() is like pouring a bunch of blocks onto the tower all at once!

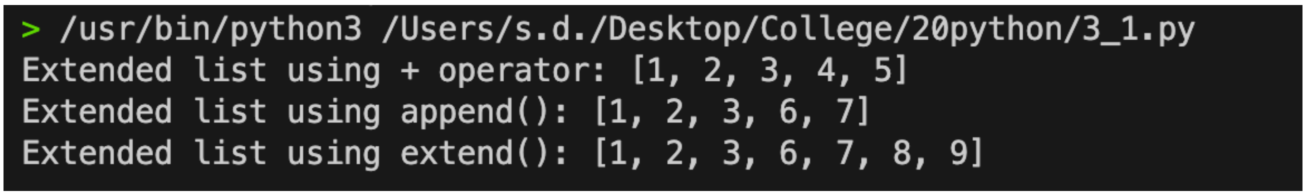
code :

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

extended = a + [4, 5]

print("Extended list using + operator:", extended)



a.append(6)

a.append(7)

print("Extended list using append():", a)
```

a.extend([8, 9])

print("Extended list using extend():", a)

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python/3_1.py
Extended list using + operator: [1, 2, 3, 4, 5]
Extended list using append(): [1, 2, 3, 6, 7]
Extended list using extend(): [1, 2, 3, 6, 7, 8, 9]
```

Conclusion : Creates a new list by concatenation, useful when preserving the original list is necessary. Efficient for adding single elements, modifying the original list in-place. Ideal for adding multiple elements from another iterable , also modifying the original list in-place.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 3.2

Title: Write a program to add two matrices.

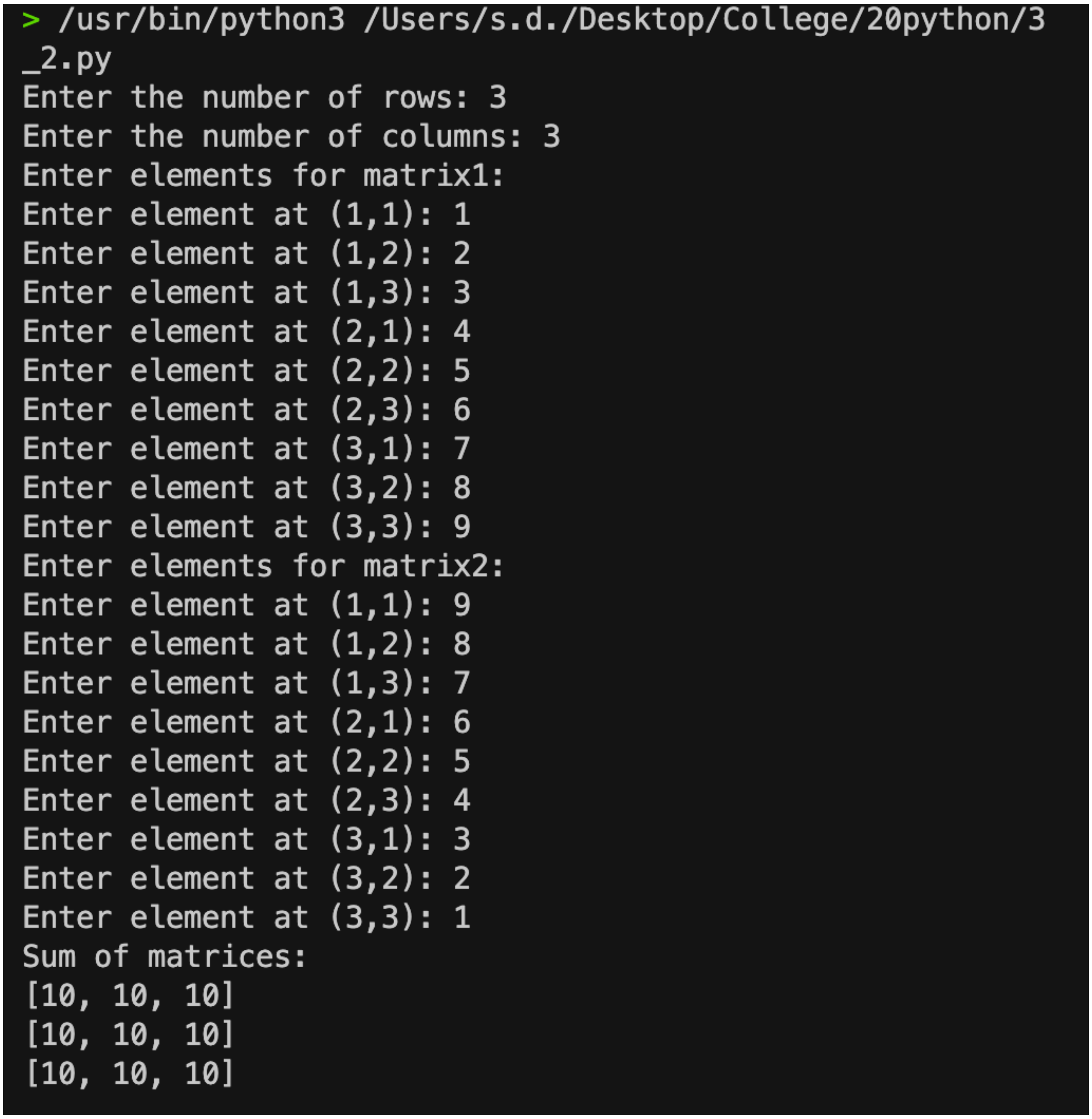
Theory : Rectangular arrays of numbers, represented as lists of lists in Python.Element-wise addition of corresponding elements in two matrices.Used to iterate through each element of the matrices.Accessing elements using row and column indices (e.g., matrix[I][j]). Checking for compatible dimensions to ensure valid matrix operations.

Code:

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

cols = int(input("Enter the number of columns: "))



print("Enter elements for matrix1:")

matrix1 = [[int(input(f"Enter element at ({i+1},{j+1}): ")) for j in range(cols)]
for i in range(rows)]



print("Enter elements for matrix2:")

matrix2 = [[int(input(f"Enter element at ({i+1},{j+1}): ")) for j in range(cols)]
for i in range(rows)]



result = [[0 for s in range(cols)] for s in range(rows)]




for i in range(rows):

  for j in range(cols):

    result[i][j] = matrix1[i][j] + matrix2[i][j]
```

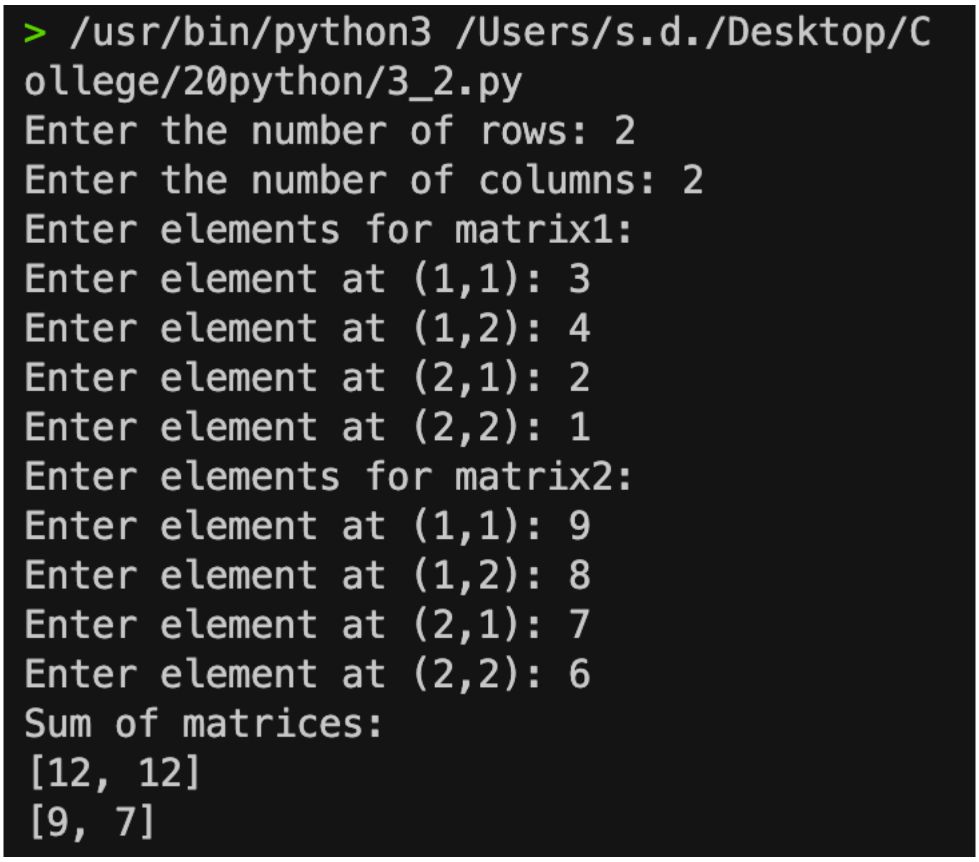
```
print("Sum of matrices:")

for row in result:

  print(row)
```

Output: (screenshot)

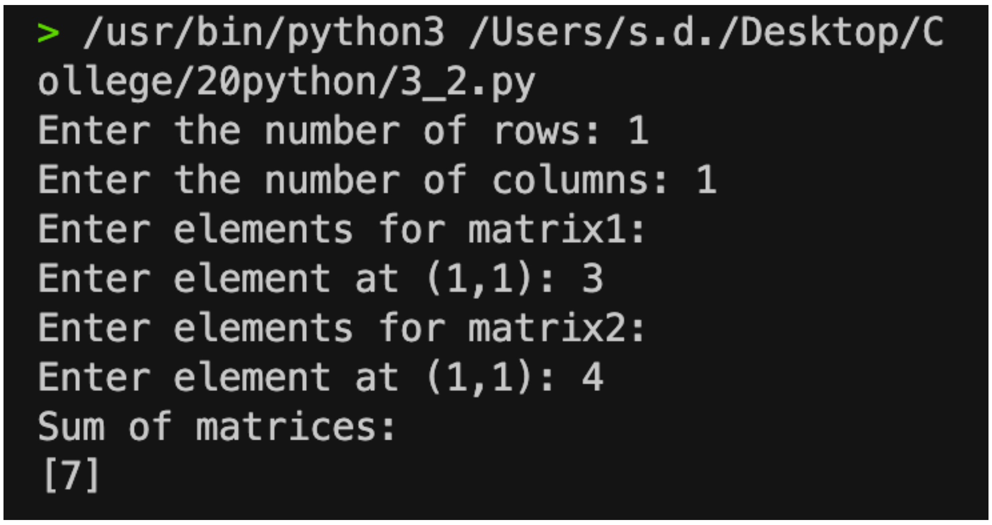


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python/3
_2.py
Enter the number of rows: 3
Enter the number of columns: 3
Enter elements for matrix1:
Enter element at (1,1): 1
Enter element at (1,2): 2
Enter element at (1,3): 3
Enter element at (2,1): 4
Enter element at (2,2): 5
Enter element at (2,3): 6
Enter element at (3,1): 7
Enter element at (3,2): 8
Enter element at (3,3): 9
Enter elements for matrix2:
Enter element at (1,1): 9
Enter element at (1,2): 8
Enter element at (1,3): 7
Enter element at (2,1): 6
Enter element at (2,2): 5
Enter element at (2,3): 4
Enter element at (3,1): 3
Enter element at (3,2): 2
Enter element at (3,3): 1
Sum of matrices:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/C
ollege/20python/3_2.py
Enter the number of rows: 2
Enter the number of columns: 2
Enter elements for matrix1:
Enter element at (1,1): 3
Enter element at (1,2): 4
Enter element at (2,1): 2
Enter element at (2,2): 1
Enter elements for matrix2:
Enter element at (1,1): 9
Enter element at (1,2): 8
Enter element at (2,1): 7
Enter element at (2,2): 6
Sum of matrices:
[12, 12]
[9, 7]
```



```
> /usr/bin/python3 /Users/s.d./Desktop/C
ollege/20python/3_2.py
Enter the number of rows: 1
Enter the number of columns: 1
Enter elements for matrix1:
Enter element at (1,1): 3
Enter elements for matrix2:
Enter element at (1,1): 4
Sum of matrices:
[7]
```

Conclusion: Python effectively performs matrix addition using nested loops and indexing. Ensuring compatible dimensions. Creating a result matrix with the same dimensions. Adding corresponding elements.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

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.

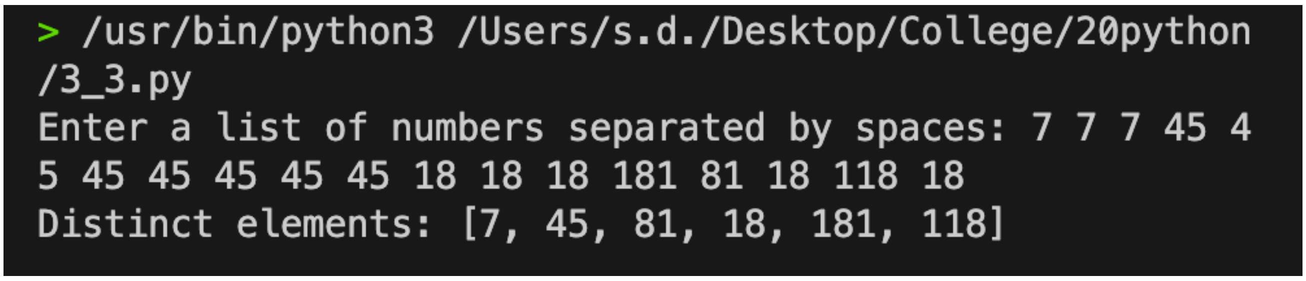
Theory : Ordered collections of items in Python, allowing duplicates . Unordered collections of unique elements.Key problem addressed by the code.Transforms user-provided string input into a list of integers.

Code :

```
def get_distinct_elements_set(input_list):

    distinct_set = set(input_list)

    return list(distinct_set)

def get_distinct_elements_loop(input_list):

  distinct_list = []

  seen = set()

  for item in input_list:

    if item not in seen:

      distinct_list.append(item)

      seen.add(item)

  return distinct_list
```

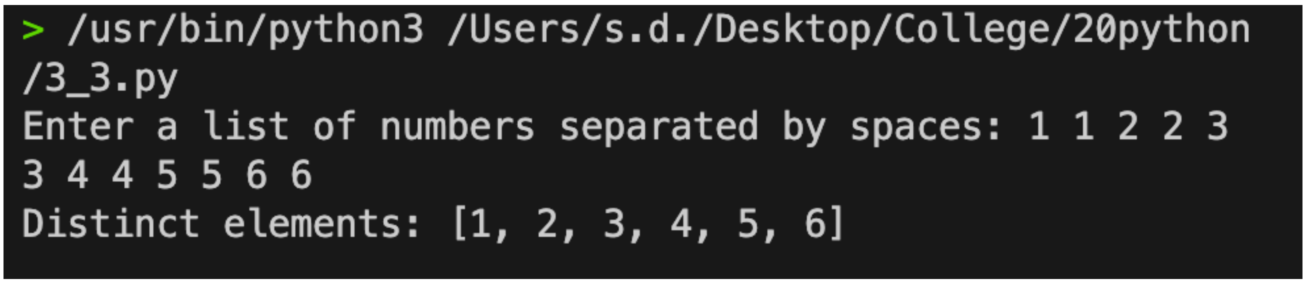
```
def get_distinct_elements_user_input():

  input_str = input("Enter a list of numbers separated by spaces: ")

  input_list = []

  for num_str in input_str.split():

    input_list.append(int(num_str))

  distinct_list = get_distinct_elements_set(input_list)

  print("Distinct elements:", distinct_list)

get_distinct_elements_user_input()
```

Output: (screenshot)

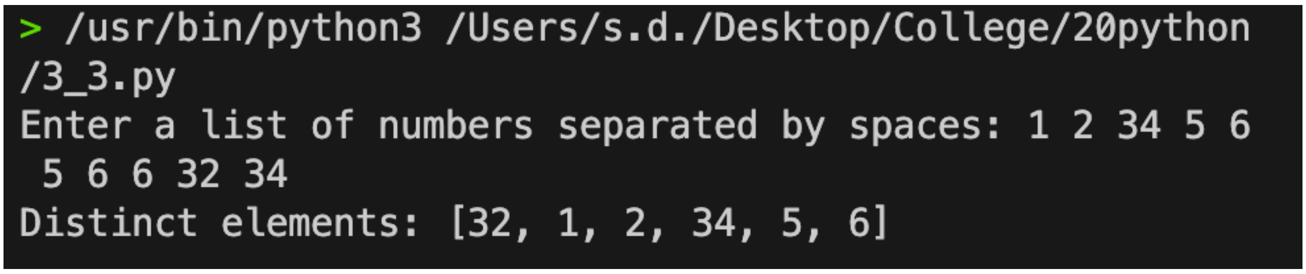


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_3.py
Enter a list of numbers separated by spaces: 7 7 7 45 4
5 45 45 45 45 45 18 18 18 181 81 18 118 18
Distinct elements: [7, 45, 81, 18, 181, 118]
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_3.py
Enter a list of numbers separated by spaces: 1 1 2 2 3
3 4 4 5 5 6 6
Distinct elements: [1, 2, 3, 4, 5, 6]
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_3.py
Enter a list of numbers separated by spaces: 1 2 34 5 6
 5 6 6 32 34
Distinct elements: [32, 1, 2, 34, 5, 6]
```

Conclusion : Python provides multiple ways to remove duplicates from a list. Set-based method is generally faster but doesn't guarantee order preservation. Loop-based method is slower but maintains order. Choice depends on efficiency and order requirements. User input allows for dynamic list creation.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 3.4

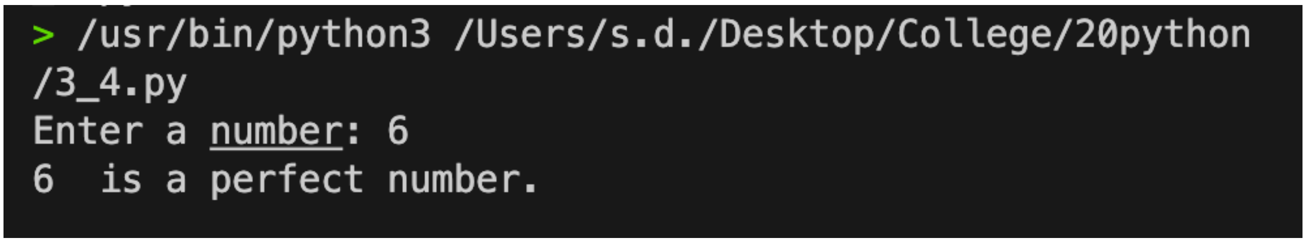
Title: Write a program to Check whether a number is perfect or not.

Theory: Numbers that equal the sum of their proper divisors (excluding themselves).Numbers that divide a given number without a remainder.The total obtained by adding all proper divisors of a number.Used to efficiently find and store divisors.Determine whether a number is perfect based on the sum of its divisors.user to provide a number and display the result.

Code:

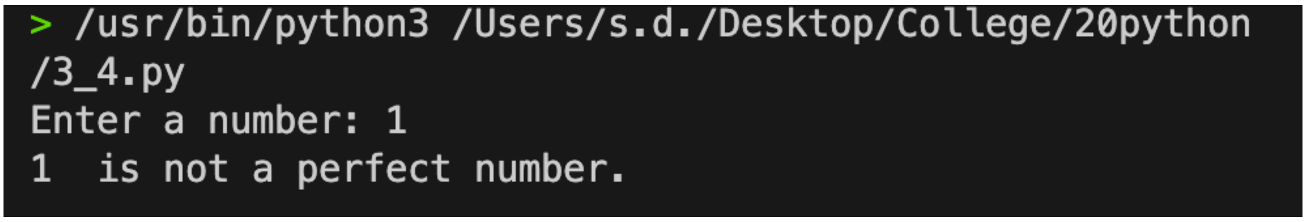
```
def a(n):

  divisors = [i for i in range(1, n) if n % i == 0]

  sum_of_divisors = sum(divisors)

  return sum_of_divisors == n

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

if a(number):

  print(number," is a perfect number.")

else:

  print(number," is not a perfect number.”)
```

Output: (screenshot)

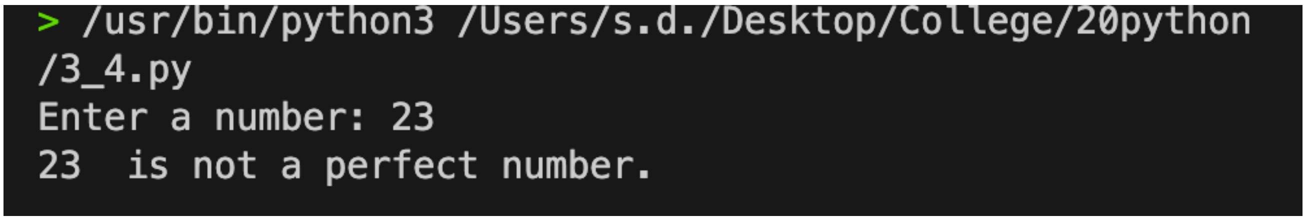


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_4.py
Enter a number: 6
6  is a perfect number.
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_4.py
Enter a number: 1
1  is not a perfect number.
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_4.py
Enter a number: 23
23  is not a perfect number.
```

Conclusion : Python effectively identifies perfect numbers using divisor calculation and comparison. Finding all proper divisors of a given number. Calculating their sum. Comparing the sum to the original number.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

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 : Sequences of characters in Python. Accessing individual characters within a string using loops. Built-in functions for string manipulation, like isupper() and islower(). Determining character case for counting. Variables used to track the occurrences of specific items. Functions can return multiple values as a tuple.

Code :

```
def count(string):

  u = 0

  l = 0

  for char in string:

    if char.isupper():

      u += 1

    elif char.islower():

      l += 1

  return u, l
```

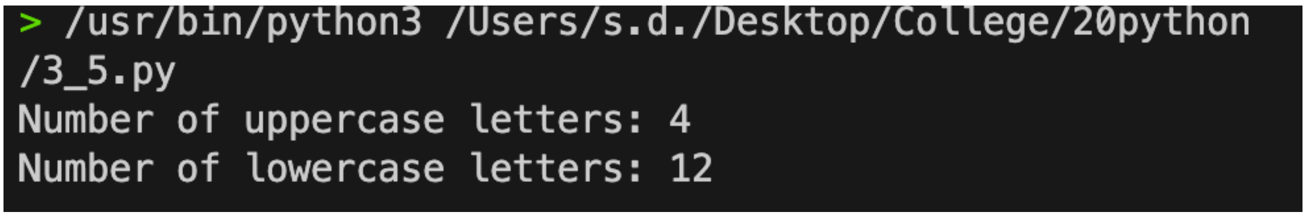
string_test = 'Today is My Best Day'

u, l = count(string_test)

print("Number of uppercase letters:", u)

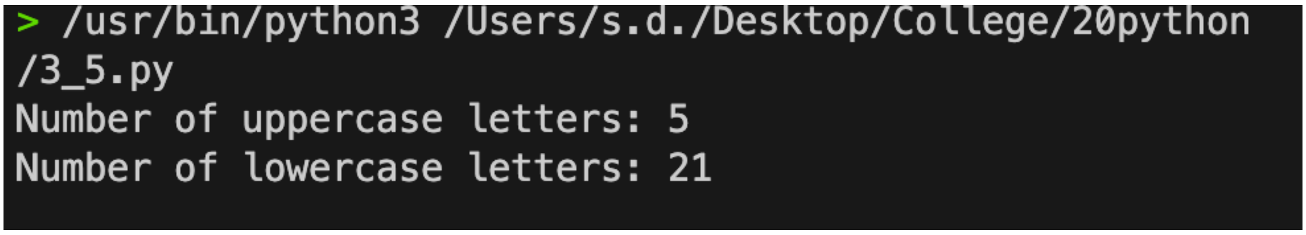
print("Number of lowercase letters:", l)

Output: (screenshot)

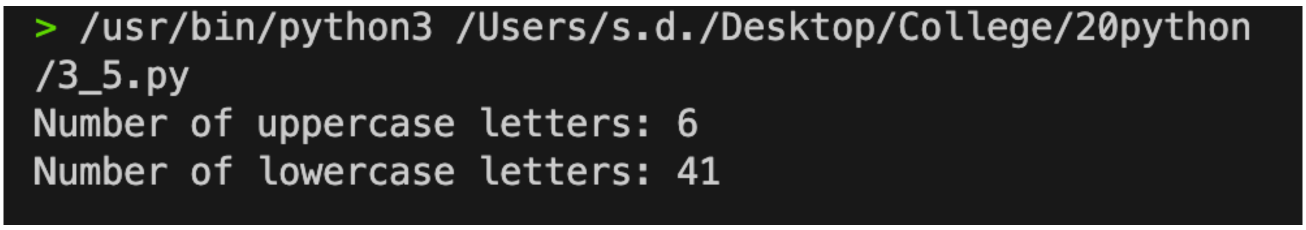


```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_5.py
Number of uppercase letters: 4
Number of lowercase letters: 12
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_5.py
Number of uppercase letters: 5
Number of lowercase letters: 21
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20python
/3_5.py
Number of uppercase letters: 6
Number of lowercase letters: 41
```

Conclusion : Python effectively counts uppercase and lowercase letters in strings using character iteration and string methods. Initializing counters for each case.Iterating through each character. Checking character case and incrementing respective counters.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4.1

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

Theory : Blueprints for creating objects with attributes (data) and methods (functions). Instances of classes, representing individual entities with specific characteristics. Variables associated with each object, storing its data. Functions associated with a class, performing actions on its objects.

Code:

```
class Employee:

    def __init__(self, name, age, department, salary)

        self.name = name

        self.age = age

        self.department = department

        self.salary = salary



    def get_details(self):



        return {

            "name": self.name,
            "age": self.age,

            "department": self.department,

            "salary": self.salary,

        }


    def print_details(self):


        print("Name:", self.name)

        print("Age: ",self.age)

        print("Department: ",self.department)

        print("Salary: $",self.salary)




emp1 = Employee("Sarthi Darji", 17, "Finance", 5000000)

emp2 = Employee("Jeevan Anaa", 19, "Marketing", 450000)

emp3=Employee("Lachh bahi",20 ,"Finance",99999)


employee1_details = emp1.get_details()

print("Employee 1 details: ",employee1_details)
```

print()

employee2_details = emp2.get_details()

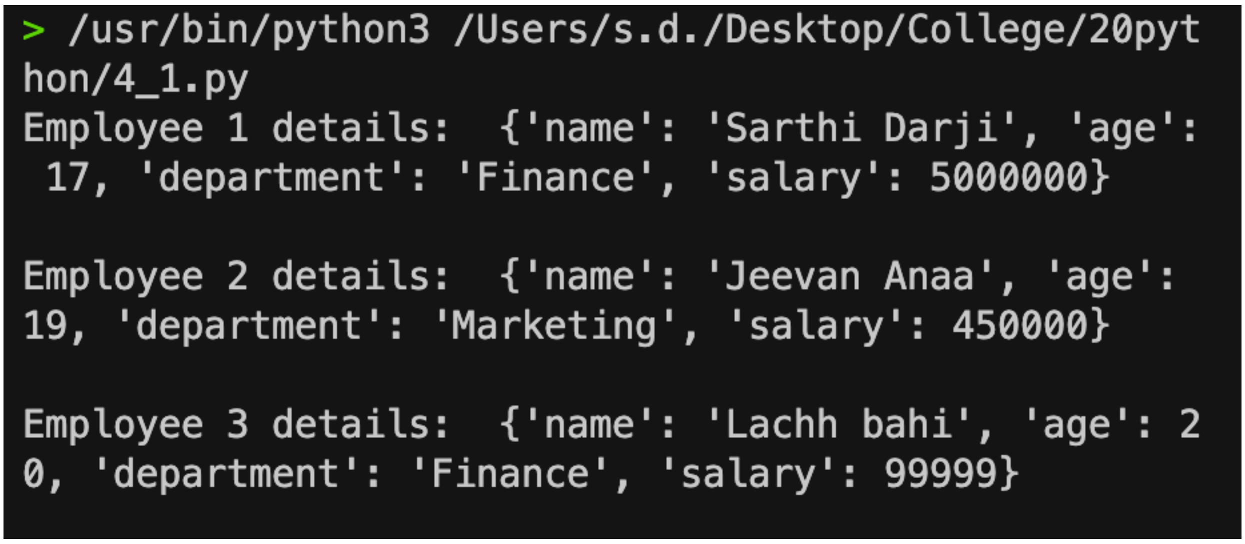
print("Employee 2 details: ",employee2_details)

print()

employee3_details = emp3.get_details()

print("Employee 3 details: “,employee3_details)

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/4_1.py
Employee 1 details:  {'name': 'Sarthi Darji', 'age':
 17, 'department': 'Finance', 'salary': 5000000}

Employee 2 details:  {'name': 'Jeevan Anaa', 'age':
19, 'department': 'Marketing', 'salary': 450000}

Employee 3 details:  {'name': 'Lachh bahi', 'age': 2
0, 'department': 'Finance', 'salary': 99999}
```

Conclusion : Creating classes to define object structure. Using constructors to set initial values. Adding methods to interact with object data.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

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 : Required arguments passed in a specific order to a function.Optional arguments passed with an identifier (name=value) to a function.*args and **kwargs enhance function versatility by allowing variable input combinations.Use dictionary-like syntax (kwargs.get(key)) to retrieve keyword arguments, defaulting to None if absent.

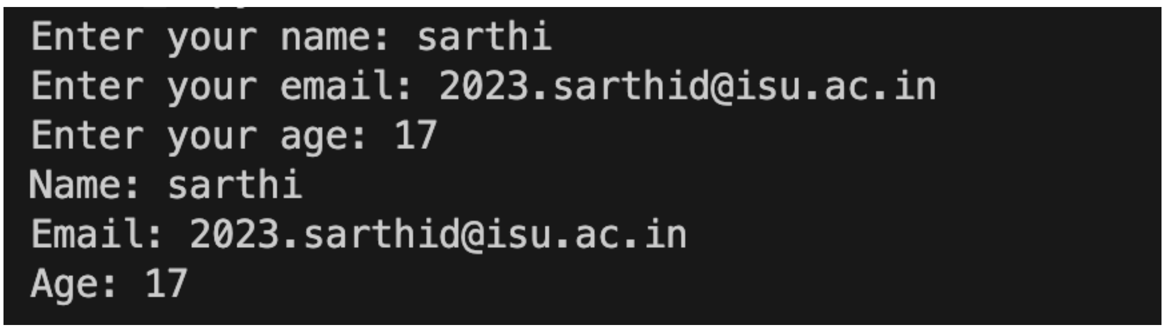
Code:

```
def get_user_info(*args, **kwargs):

    name = args[0]

    email = args[1]

    age = kwargs.get("age")

    return name, email, age

name = input("Enter your name: ")

email = input("Enter your email: ")

age = input("Enter your age: ")

name, email, age = get_user_info(name, email, age=age)

print("Name:", name)
```

print("Email:", email)

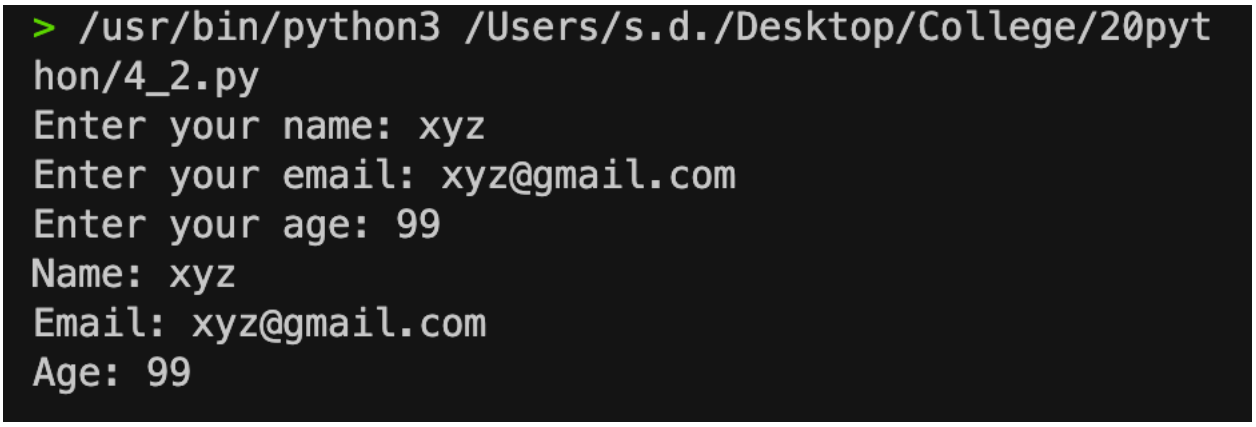
print("Age:", age)

Output: (screenshot)

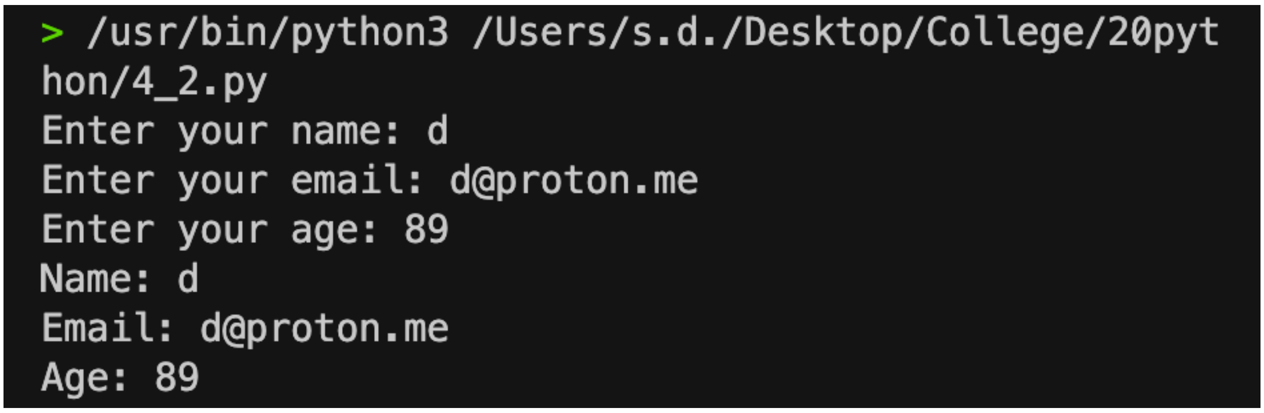


```
Enter your name: sarthi
Enter your email: 2023.sarthid@isu.ac.in
Enter your age: 17
Name: sarthi
Email: 2023.sarthid@isu.ac.in
Age: 17
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/4_2.py
Enter your name: xyz
Enter your email: xyz@gmail.com
Enter your age: 99
Name: xyz
Email: xyz@gmail.com
Age: 99
```



```
> /usr/bin/python3 /Users/s.d./Desktop/College/20pyt
hon/4_2.py
Enter your name: d
Enter your email: d@proton.me
Enter your age: 89
Name: d
Email: d@proton.me
Age: 89
```

Conclusion : Python's *args and **kwargs facilitate flexible function argument handling. Use *args for flexible positional argument collection. Use **kwargs for flexible keyword argument collection. Employ kwargs.get(key) for safe keyword argument retrieval.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4.6

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

Theory: It's like mixing super-secret formulas to create a new one! The DerivedClass borrows special ingredients (attributes and methods) from both LU and ITM classes, adding its own unique flavor.It's a powerful way to build complex creations from simpler building blocks, like mixing colors to paint a masterpiece.

Code :

```
class LU:

    def __init__(self, lu_code, lu_name):

        self.lu_code = lu_code

        self.lu_name = lu_name



    def display_lu_info(self):

        print("LU Code:", self.lu_code)

        print("LU Name:", self.lu_name)
```

```
class ITM:

    def __init__(self, itm_code, itm_name):

        self.itm_code = itm_code

        self.itm_name = itm_name



    def display_itm_info(self):

        print("ITM Code:", self.itm_code)

        print(f"ITM Name:", self.itm_name)

class DerivedClass(LU, ITM):

    def __init__(self, lu_code, lu_name, itm_code, itm_name, derived_info):

        LU.__init__(self, lu_code, lu_name)

        ITM.__init__(self, itm_code, itm_name)

        self.derived_info = derived_info



    def display_derived_info(self):

        print("Information:", self.derived_info)

lu_code = "001"

lu_name = "Marketing"

itm_code = "001"
```

itm_name = "Finance"

derived_info = "Thank you.."

derived_object = DerivedClass(lu_code, lu_name, itm_code, itm_name, derived_info)

derived_object.display_lu_info()

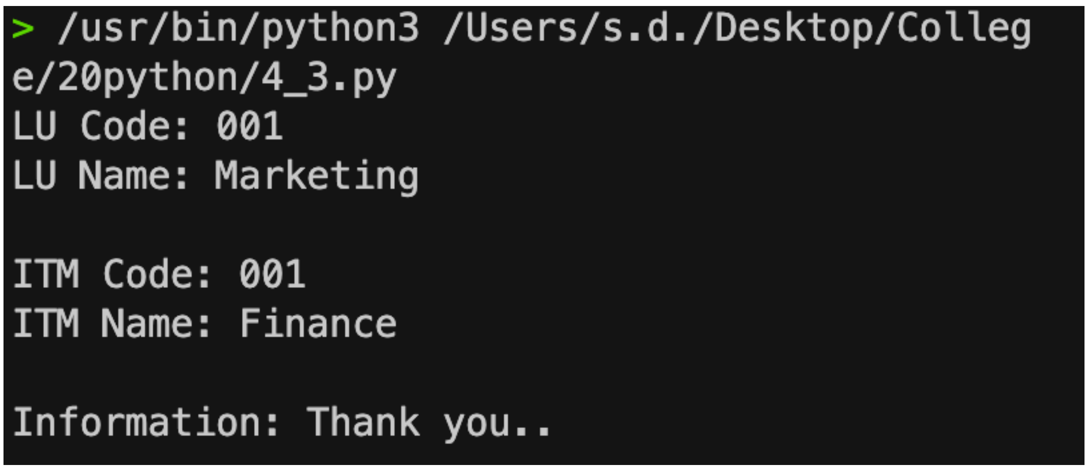
print()

derived_object.display_itm_info()

print()

derived_object.display_derived_info()

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_3.py
LU Code: 001
LU Name: Marketing

ITM Code: 001
ITM Name: Finance

Information: Thank you..
```

Conclusion : The code demonstrates multiple inheritance, creating a DerivedClass that inherits from both LU and ITM classes.It showcases how a derived class can combine and extend the functionality of its base classes.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4.4

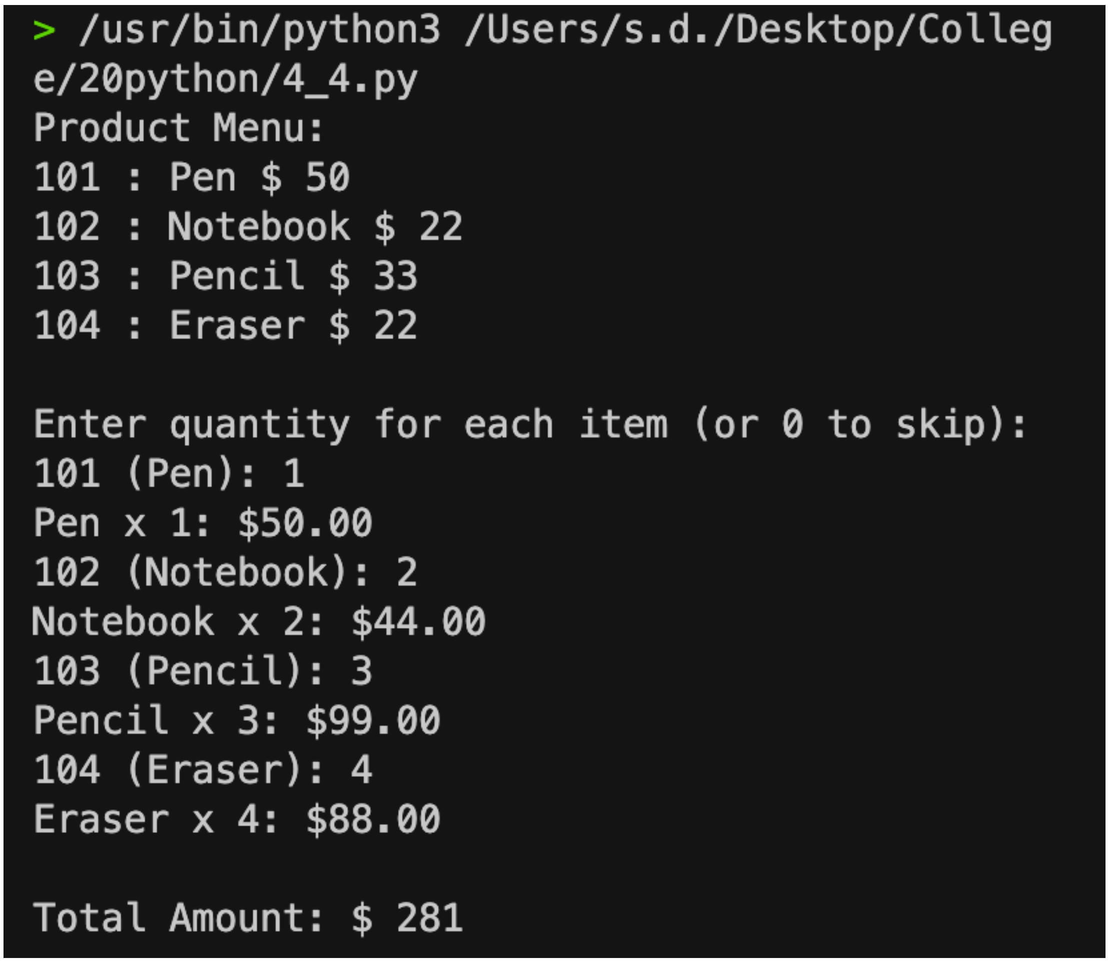
Title: Write a program that has a class store which keeps the record of code and price of product display the menu of all product and prompt to enter the quantity ofeach item required and finally generate the bill and display the total amount

Theory: It's like building a mini shopping assistant! The code uses blueprints called classes to create a store with special powers.It stores product secrets in a magic dictionary.It talks to shoppers to know what they want.It does fancy math to figure out the bill.It prints a shopping summary like a pro!

Code :

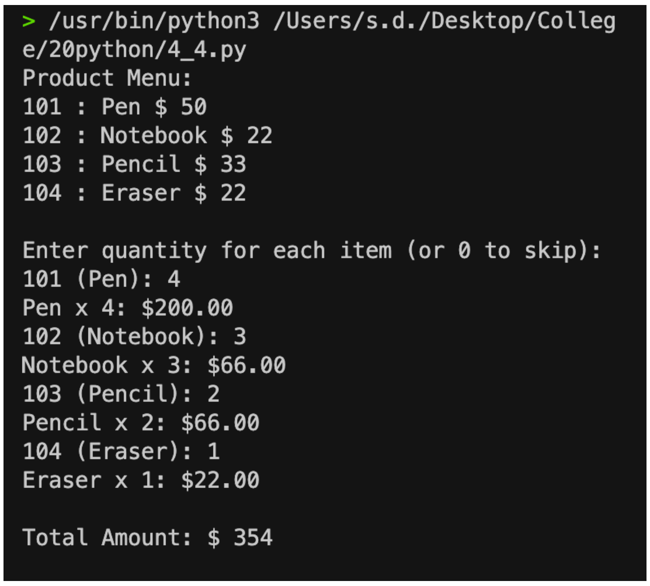
```
class Store:

    def __init__(self):

        self.products = {

            "101": {"name": "Pen", "price": 50},

            "102": {"name": "Notebook", "price":22},

            "103": {"name": "Pencil", "price": 33},
            "104": {"name": "Eraser", "price": 22},

        }

    def display_menu(self):

        print("Product Menu:")

        for code, product in self.products.items():

            print(code ,”:",product['name'],"$",product['price'])

   def generate_bill(self):

        total_amount = 0

        print("\nEnter quantity for each item (or 0 to skip):")

        for code, product in self.products.items():

            quantity = int(input(f"{code} ({product['name']}): "))

            if quantity > 0:

                item_amount = product["price"] * quantity

                print(f"{product['name']} x {quantity}: ${item_amount:.2f}")

                total_amount += item_amount

        print("\nTotal Amount: $",total_amount)

store = Store()

store.display_menu()

store.generate_bill()
```

Output: (screenshot)

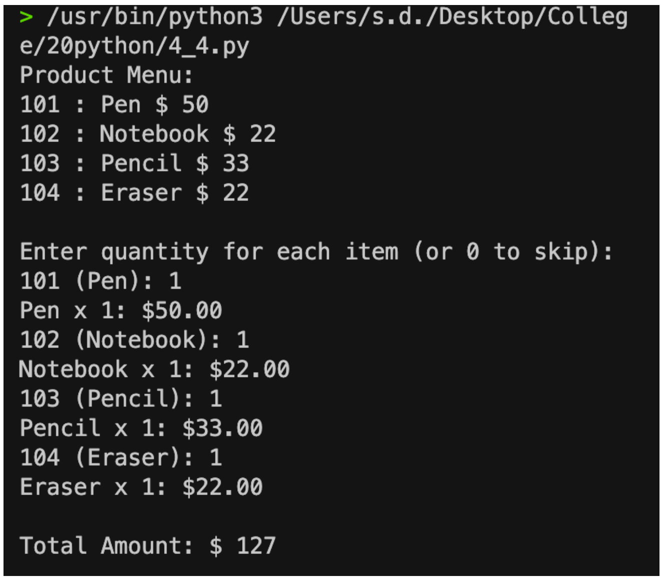


```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_4.py
Product Menu:
101 : Pen $ 50
102 : Notebook $ 22
103 : Pencil $ 33
104 : Eraser $ 22

Enter quantity for each item (or 0 to skip):
101 (Pen): 1
Pen x 1: $50.00
102 (Notebook): 2
Notebook x 2: $44.00
103 (Pencil): 3
Pencil x 3: $99.00
104 (Eraser): 4
Eraser x 4: $88.00

Total Amount: $ 281
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_4.py
Product Menu:
101 : Pen $ 50
102 : Notebook $ 22
103 : Pencil $ 33
104 : Eraser $ 22

Enter quantity for each item (or 0 to skip):
101 (Pen): 4
Pen x 4: $200.00
102 (Notebook): 3
Notebook x 3: $66.00
103 (Pencil): 2
Pencil x 2: $66.00
104 (Eraser): 1
Eraser x 1: $22.00

Total Amount: $ 354
```



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_4.py
Product Menu:
101 : Pen $ 50
102 : Notebook $ 22
103 : Pencil $ 33
104 : Eraser $ 22

Enter quantity for each item (or 0 to skip):
101 (Pen): 1
Pen x 1: $50.00
102 (Notebook): 1
Notebook x 1: $22.00
103 (Pencil): 1
Pencil x 1: $33.00
104 (Eraser): 1
Eraser x 1: $22.00

Total Amount: $ 127
```

Conclusion : The code effectively simulates a store experience with product browsing and bill generation.It showcases key Python concepts like classes, dictionaries, user input, loops, and string formatting.It demonstrates object-oriented programming principles for code organization and reusability.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4.5

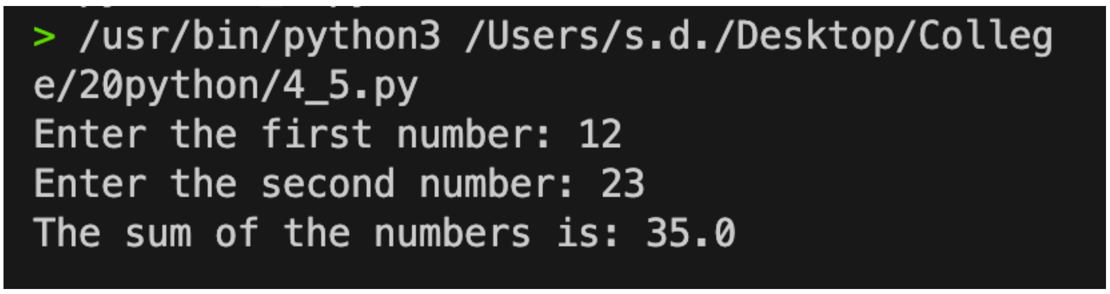
Title: Write a program to take input from user for addition of two numbers using (single inheritance).

Theory : The Adder class extends the Number class, establishing a "is-a" relationship. This implies the Adder inherits all attributes and methods defined in Number unless explicitly overridden.

Code :

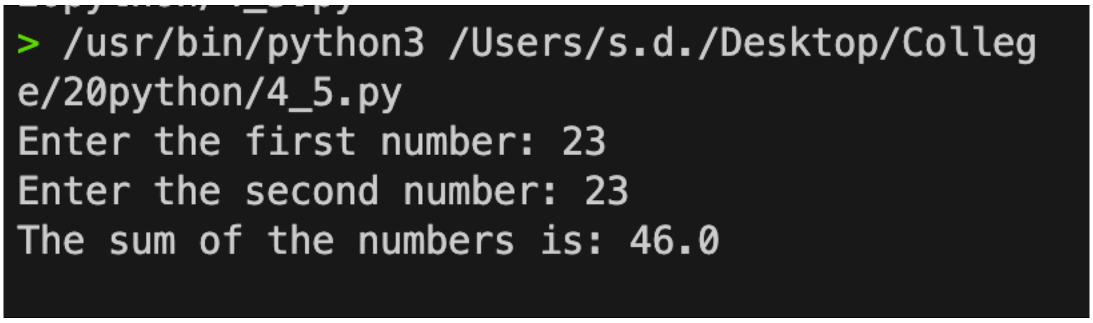
```
class Number:

    def get_numbers(self):

        num1 = float(input("Enter the first number: "))

        num2 = float(input("Enter the second number: "))

        return num1, num2

class Adder(Number):

    def add(self):
        num1, num2 = self.get_numbers()

        result = num1 + num2

        print("The sum of the numbers is:", result)

adder = Adder()

adder.add()
```

Output: (screenshot)

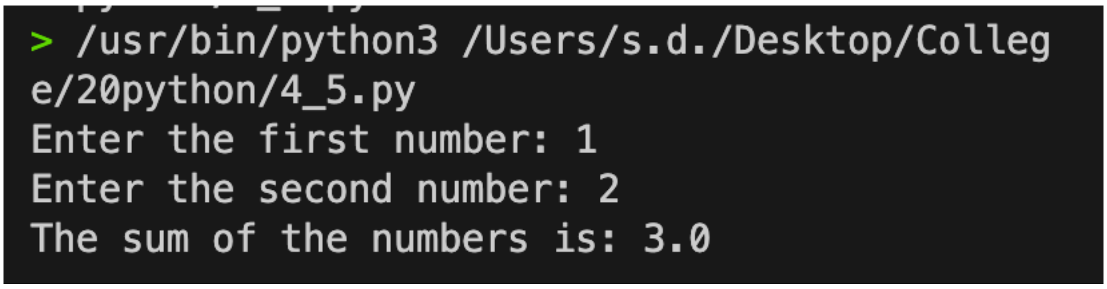


```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_5.py
Enter the first number: 12
Enter the second number: 23
The sum of the numbers is: 35.0
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_5.py
Enter the first number: 23
Enter the second number: 23
The sum of the numbers is: 46.0
```



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_5.py
Enter the first number: 1
Enter the second number: 2
The sum of the numbers is: 3.0
```

Conclusion : This Python program showcases single inheritance to build a simple adder program.It demonstrates how inheritance simplifies code structure and promotes good programming practices.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4.3

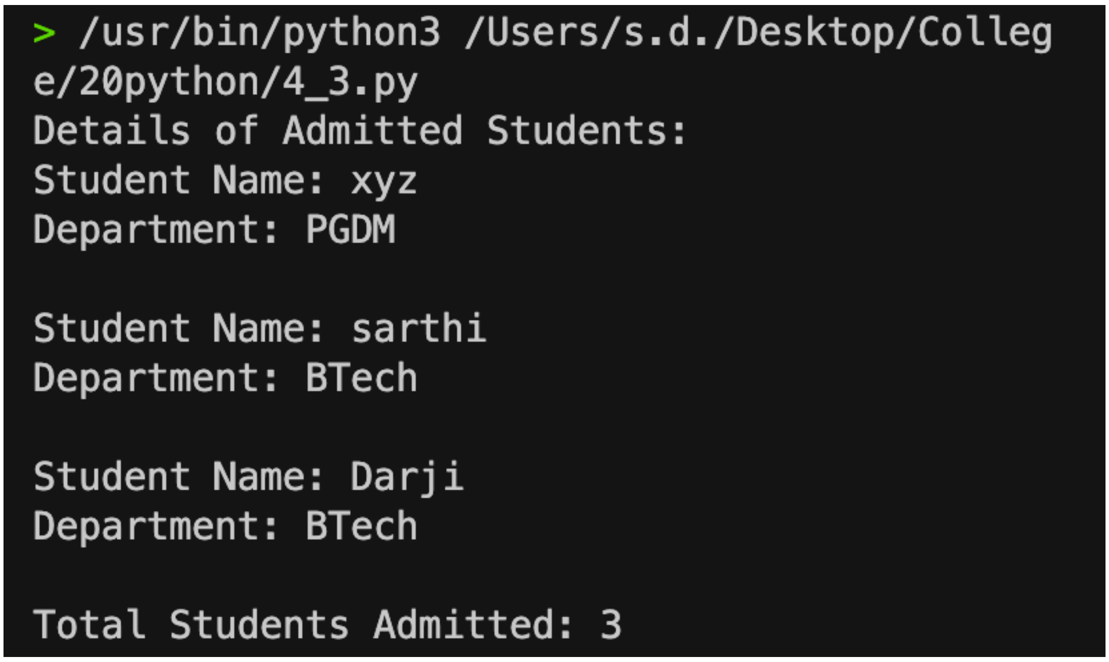
Title: Write a program to admit the students in the different Departments(pgdm/btech)and count the students. (Class, Object and Constructor).

Theory : Classes act as blueprints for creating student profiles. Constructors craft individualized student objects. Instance variables hold unique student details. Class variables track collective student information. Methods enable interaction with student data.

Code :

```
class Student:

    total_students = 0

    def __init__(self, name, department):

        self.name = name

        self.department = department

        Student.total_students += 1

    def display_details(self):
        print(f"Student Name: {self.name}")

        print(f"Department: {self.department}")

        print()

# Example usage:

# Create instances of Student class for different students

student1 = Student(name="xyz ", department="PGDM")

student2 = Student(name="sarthi", department="BTech")

student3 = Student(name="Darji", department="BTech")


# Display details of each student

print("Details of Admitted Students:")

student1.display_details()

student2.display_details()

student3.display_details()

# Display the total number of students

print(f"Total Students Admitted: {Student.total_students}”)
```

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_3.py
Details of Admitted Students:
Student Name: xyz
Department: PGDM

Student Name: sarthi
Department: BTech

Student Name: Darji
Department: BTech

Total Students Admitted: 3
```

Conclusion : The code effectively demonstrates object-oriented programming concepts for representing student information. It highlights the use of constructors, instance variables, class variables, and methods for data organization and access.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4.7

Title: Write a program to implement Multilevel inheritance,

Grandfather⍰Father-⍰Child to show property inheritance from grandfather to child.

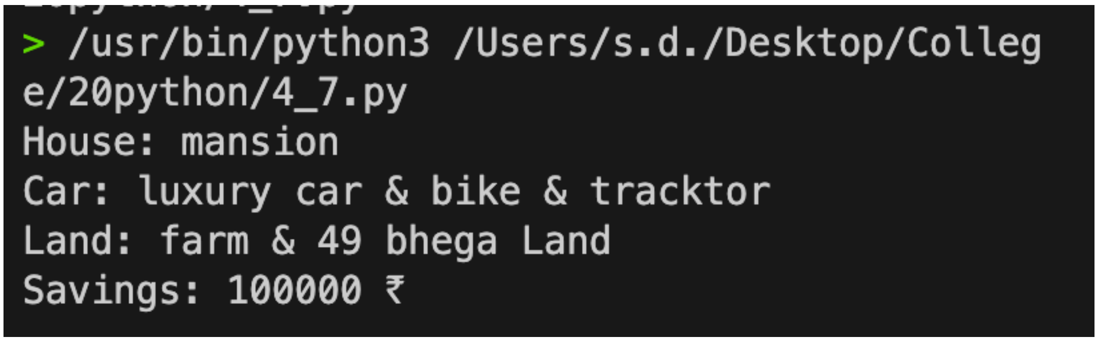
Theory : Classes act as blueprints for creating family members with their respective assets. Multilevel inheritance allows assets to flow from grandparents to parents to children. Constructors ensure proper initialization of assets for each object**.**

Code :

```
class Grandfather:

    def __init__(self, house, car):

        self.house = house

        self.car = car



class Father(Grandfather):

    def __init__(self, house, car, land):

        super().__init__(house, car)

        self.land = land



class Child(Father):

    def __init__(self, house, car, land, savings):

        super().__init__(house, car, land)

        self.savings = savings

child = Child("mansion", "luxury car & bike & tractor", "farm & 49 bhega 
Land", 100000)
```

```
# Access inherited assets

print(f"House: {child.house}")

print(f"Car: {child.car}")

print(f"Land: {child.land}")

print(f"Savings: {child.savings}”,”₹")
```

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_7.py
House: mansion
Car: luxury car & bike & tracktor
Land: farm & 49 bhega Land
Savings: 100000 ₹
```

Conclusion : The code effectively illustrates how multilevel inheritance can be used to model real-world concepts like asset inheritance across generations. It highlights the benefits of code reuse and organization through inheritance, making it easier to manage and track assets in a structured way.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

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)

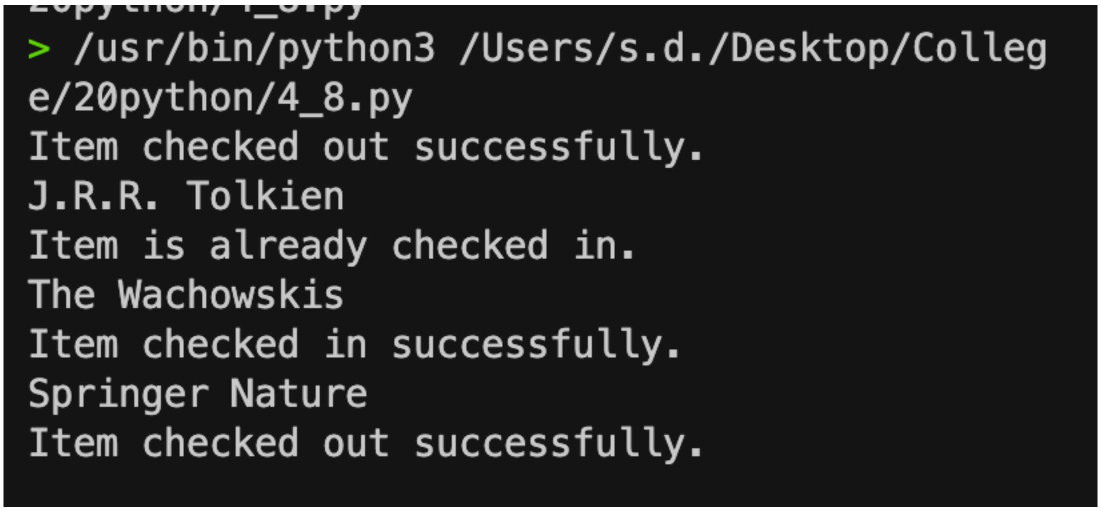
Theory : The code models a library catalog using a base class (LibraryItem) and derived classes (Book, DVD, Journal) to manage different item types efficiently. Common attributes and methods are defined in the base class, while unique attributes are added in derived classes to capture specific details for each item type.

Code :

```
class LibraryItem:

    def __init__(self, title, author):

        self.title = title

        self.author = author

        self.is_checked_out = False



    def check_out(self):

        if self.is_checked_out:

            print("Item is already checked out.")
        else:
            self.is_checked_out = True
            print("Item checked out successfully.")

    def check_in(self):
        if not self.is_checked_out:
            print("Item is already checked in.")
        else:
            self.is_checked_out = False
            print("Item checked in successfully.")

class Book(LibraryItem):
    def __init__(self, title, author, isbn):
        super().__init__(title, author)
        self.isbn = isbn

class DVD(LibraryItem):
    def __init__(self, title, director, release_year):
        super().__init__(title, director)
        self.release_year = release_year


class Journal(LibraryItem):

    def __init__(self, title, publisher, issue_number):

        super().__init__(title, publisher)

        self.issue_number = issue_number


# Example usage:

book = Book("The Lord of the Rings", "J.R.R. Tolkien", "9780547928227")

dvd = DVD("The Matrix", "The Wachowskis", 1999)

journal = Journal("Nature", "Springer Nature", 554)


# Check out items

book.check_out()

print(book.author)

dvd.check_in()

print(dvd.author)

# Check in items

book.check_in()
```

```
print(journal.author)

journal.check_out()
```

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/4_8.py
Item checked out successfully.
J.R.R. Tolkien
Item is already checked in.
The Wachowskis
Item checked in successfully.
Springer Nature
Item checked out successfully.
```

Conclusion : The code demonstrates the power of inheritance for designing organized and adaptable library systems. It effectively models different item types with shared and unique characteristics, promoting code reusability and flexibility.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 5.1

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

Theory : Python encourages modularity, breaking code into self-contained modules for better organization, reusability, and maintainability. A module is a Python file containing functions, variables, and other code elements. The import statement allows code from one module to be used in another, promoting code sharing and collaboration.

Code :

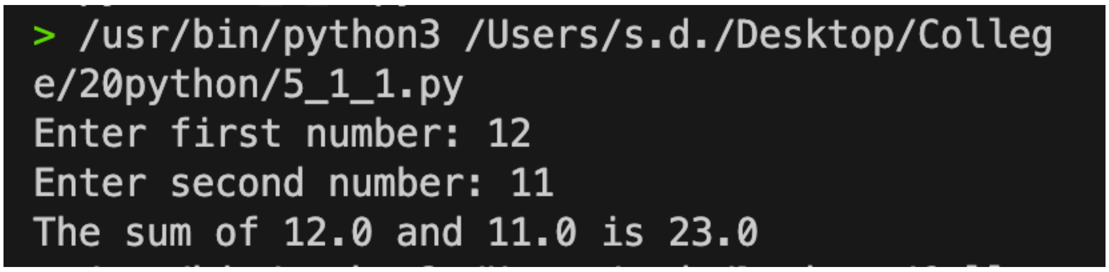
My mod

```
def add_numbers(num1, num2):

  sum = num1 + num2

  return sum
```

5_1_1.py

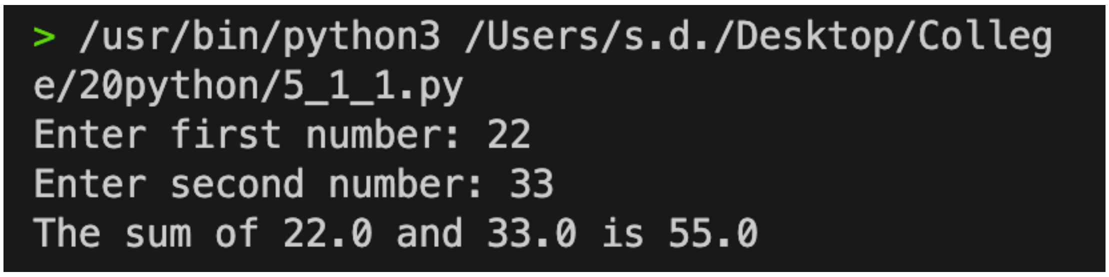
```
import mymod

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

result = mymod.add_numbers(num1, num2)

print("The sum of", num1, "and", num2, "is", result)
```

Output: (screenshot)

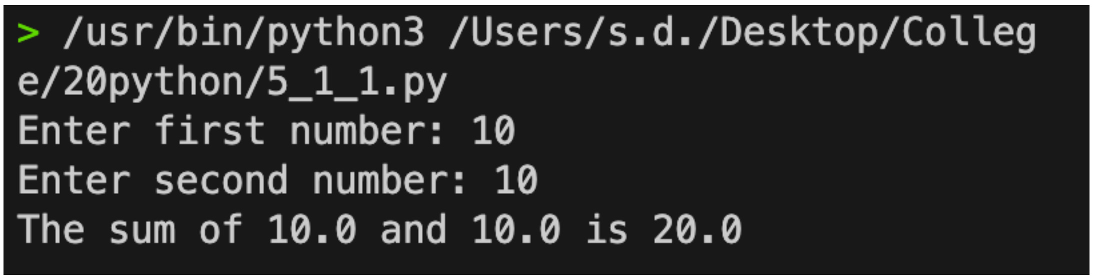


```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/5_1_1.py
Enter first number: 12
Enter second number: 11
The sum of 12.0 and 11.0 is 23.0
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/5_1_1.py
Enter first number: 22
Enter second number: 33
The sum of 22.0 and 33.0 is 55.0
```



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/5_1_1.py
Enter first number: 10
Enter second number: 10
The sum of 10.0 and 10.0 is 20.0
```

Conclusion : The code demonstrates the creation and usage of custom modules in Python. It highlights the benefits of modular programming for code organization and reusability.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

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 : Separating bank functionality into a module promotes code organization and reusability.The BankAccount class encapsulates account data and actions, modeling real-world banking concepts. The main file provides a menu-driven interface with clear choices and prompts for user interaction. User inputs are validated to prevent errors and ensure data integrity.

Code :

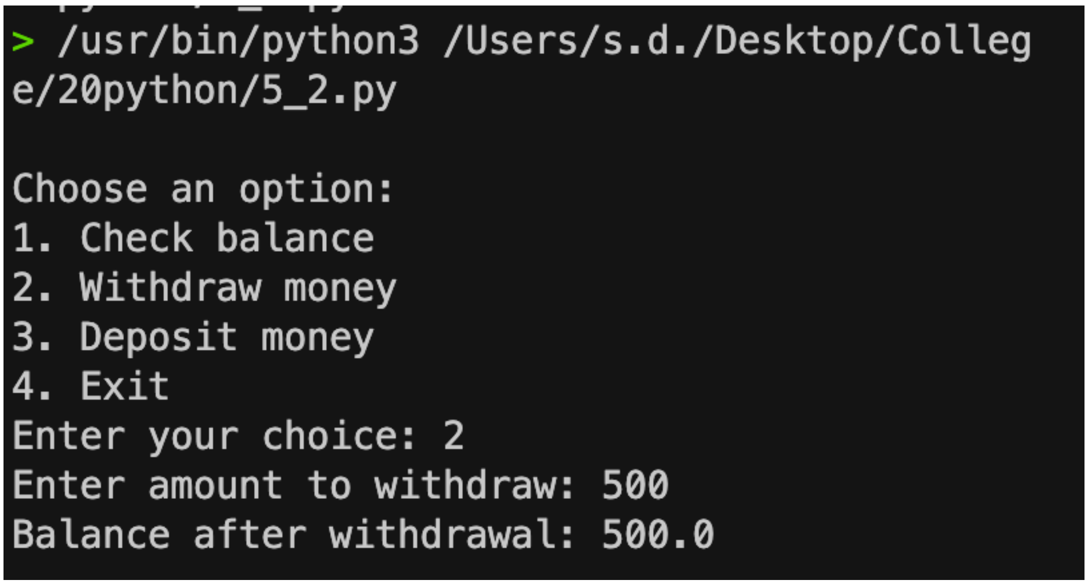
Bank_mod.py

```
class BankAccount:

    def __init__(self, account_number, balance):

        self.account_number = account_number

        self.balance = balance

    def check_balance(self):

        return self.balance
    def withdraw(self, amount):

        if amount > self.balance:

            raise ValueError("Insufficient funds")

        self.balance -= amount

        return self.balance

    def deposit(self, amount):

        if amount < 0:

            raise ValueError("Deposit amount cannot be negative")

        self.balance += amount

        return self.balance
```

5_2.py

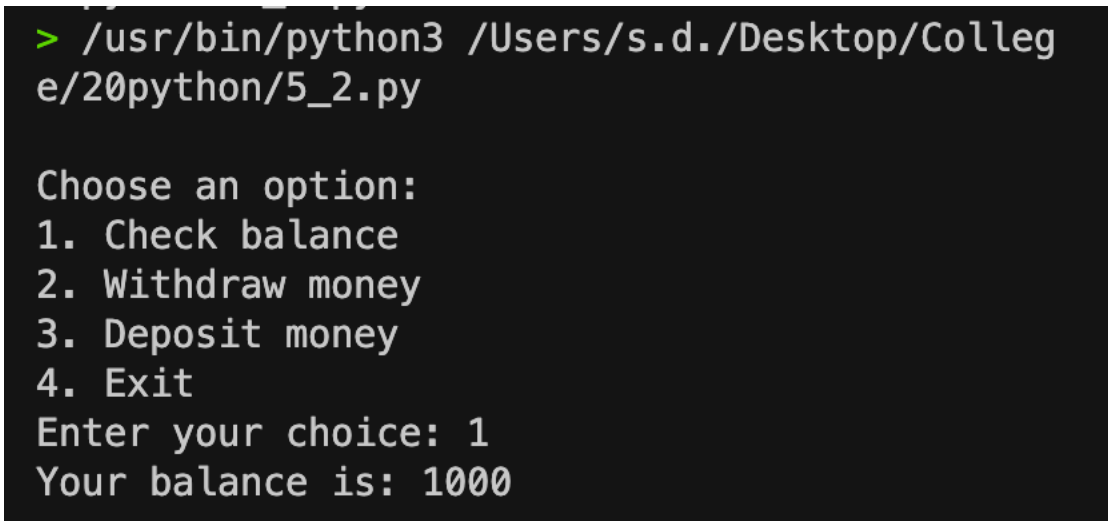
```
import Bank_mod

account1 =Bank_mod.BankAccount(12345, 1000)

while True:

    print("\nChoose an option:")

    print("1. Check balance")

    print("2. Withdraw money")
    print("3. Deposit money")

    print("4. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":

        balance = account1.check_balance()

        print("Your balance is:", balance)

    elif choice == "2":

        amount = float(input("Enter amount to withdraw: "))

        try:

            balance = account1.withdraw(amount)

            print("Balance after withdrawal:", balance)

        except ValueError as e:

            print(e)

    elif choice == "3":

        amount = float(input("Enter amount to deposit: "))

        try:

            balance = account1.deposit(amount)

            print("Balance after deposit:", balance)

        except ValueError as e:
            print(e)

    elif choice == "4":

        print("Exiting...")

        break

    else:

        print("Invalid choice. Please try again.”)
```

Output: (screenshot)

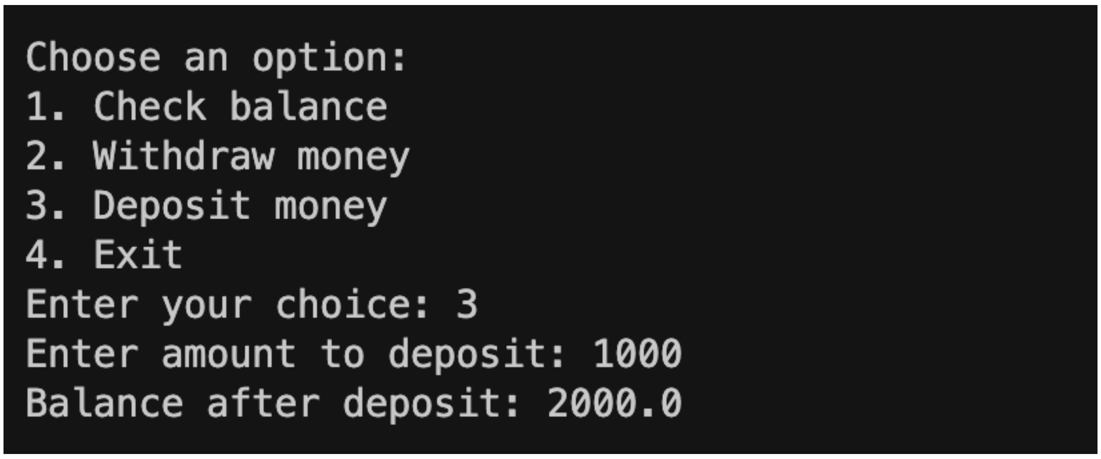


```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/5_2.py

Choose an option:
1. Check balance
2. Withdraw money
3. Deposit money
4. Exit
Enter your choice: 2
Enter amount to withdraw: 500
Balance after withdrawal: 500.0
```

Test Case: Any two (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/5_2.py

Choose an option:
1. Check balance
2. Withdraw money
3. Deposit money
4. Exit
Enter your choice: 1
Your balance is: 1000
```



```
Choose an option:
1. Check balance
2. Withdraw money
3. Deposit money
4. Exit
Enter your choice: 3
Enter amount to deposit: 1000
Balance after deposit: 2000.0
```

Conclusion : The code effectively demonstrates modular programming, object-oriented principles, and user-centric design to create an interactive and adaptable bank application. It lays a foundation for building more complex banking systems with enhanced features and security measures.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

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 : Packages group related modules together, structuring large codebases for better maintainability and navigation. Modules encapsulate code with specific functionality, promoting code reuse and reducing redundancy. Classes define blueprints for objects, encapsulating data (attributes) and actions (methods) to model real-world entities like cars. Methods are invoked on objects to perform actions, allowing interaction with the objects' state and capabilities

Code:

BMW.py

```
class BMW:

    def __init__(self, model, color):

        self.model = model

        self.color = color

    def start(self):

        print("BMW starting...")

    def accelerate(self):

        print("BMW accelerating…")
```

AUDI.py

```
class AUDI:

    def __init__(self, model, horsepower):

        self.model = model

        self.horsepower = horsepower

   def drive(self):
        print(f"AUDI {self.model} driving with {self.horsepower} horsepower.")
```

NISSAN.py

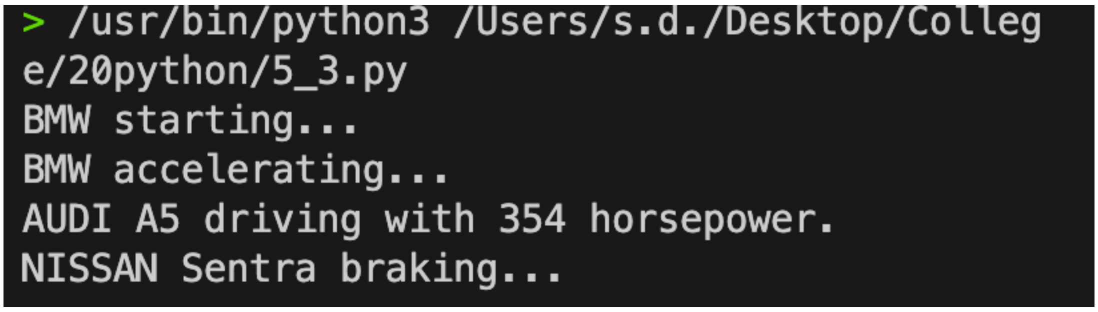
```
class NISSAN:
    def __init__(self, model, fuel_type):
        self.model = model
        self.fuel_type = fuel_type
    def brake(self):
        print(f"NISSAN {self.model} braking…")
```

5_3.py

```
from cars.BMW import BMW
from cars.AUDI import AUDI
from cars.NISSAN import  NISSAN
bmw = BMW("M3", "blue")
audi = AUDI("A5", 354)
nissan = NISSAN("Sentra", "hybrid")
bmw.start()
bmw.accelerate()
audi.drive()
```

```
nissan.brake()
```

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/5_3.py
BMW starting...
BMW accelerating...
AUDI A5 driving with 354 horsepower.
NISSAN Sentra braking...
```

Conclusion : The code effectively demonstrates how packages, modules, and classes work together to create organized, reusable, and object-oriented Python applications. It highlights the importance of choosing appropriate import strategies to access and utilize code components correctly.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 6.1

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

Theory : Threads enable concurrent execution within a process. threading module handles thread management. Thread objects represent individual threads. start() launches thread execution. join() waits for thread completion**.**

Code :

```
import threading



def print_hello():

    print("Hello")



def print_hi():

    print("Hi")



# Create threads

thread1 = threading.Thread(target=print_hello)

thread2 = threading.Thread(target=print_hi)
```

# Start threads

thread1.start()

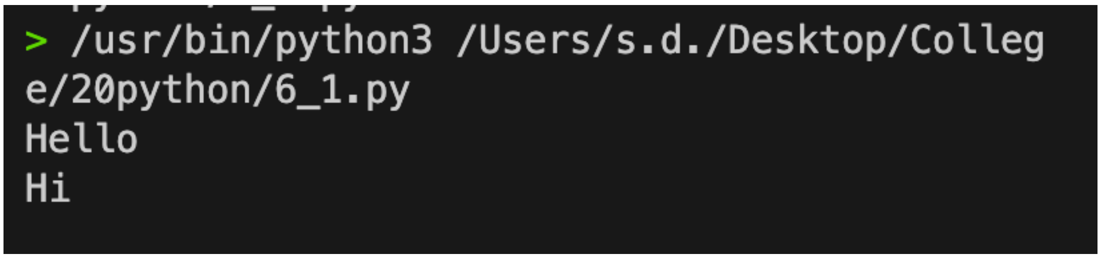
thread2.start()

# Wait for threads to finish (optional, but recommended for synchronization)

thread1.join()

thread2.join()

Output: (screenshot)



```
> /usr/bin/python3 /Users/s.d./Desktop/Colleg
e/20python/6_1.py
Hello
Hi
```

Conclusion : The code demonstrates fundamental multithreading concepts in Python, showcasing how to create and run multiple threads concurrently. It highlights the use of threading.Thread objects, start() method for execution, and join() method for synchronization.t illustrates the non-deterministic nature of thread execution, emphasizing the importance of synchronization mechanisms when precise order is required.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 7.1

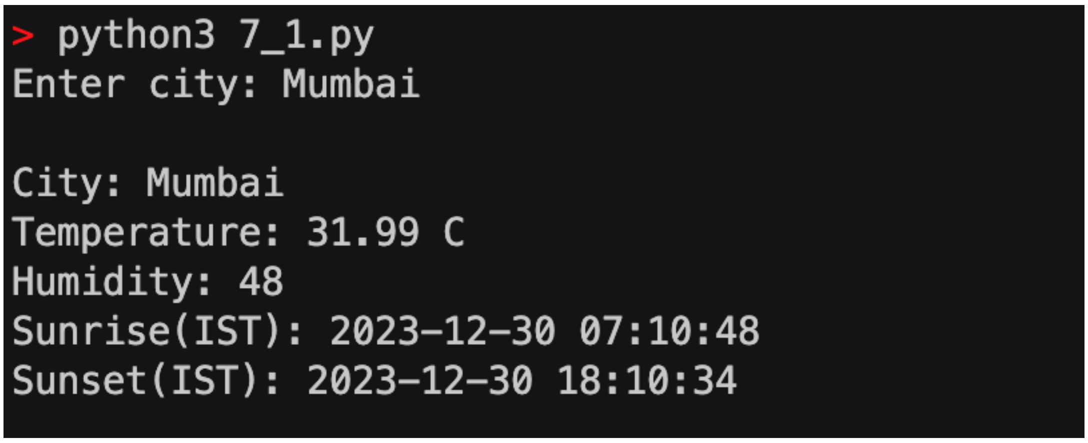
Title: Write a program to use ‘whether API’ and print temperature of any city, also print the sunrise and sunset times for the same humidity of that area.

Theory : Web APIs offer rich data sources for applications. requests simplifies API interactions. JSON is a common API data format. API keys often control API access. Error handling ensures robustness. datetime handles date and time operations.

Code :

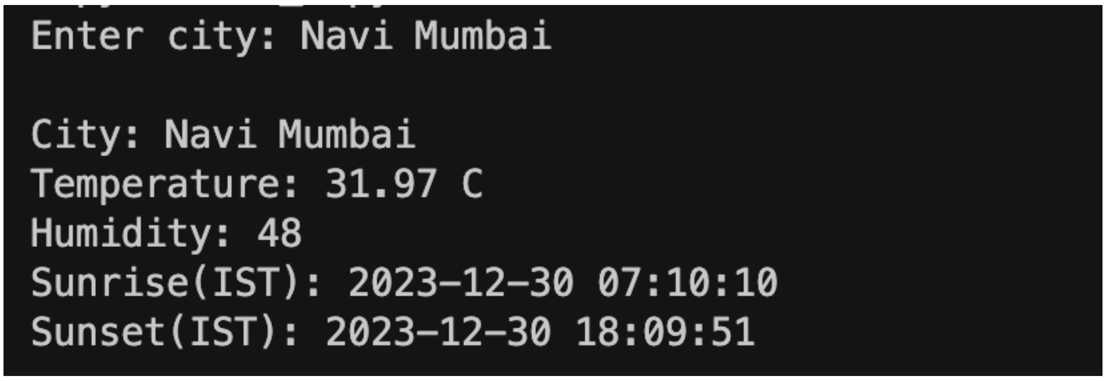
```
api_key="6126a5d84240bd19cddc584c3a690750"

import requests

import datetime

city=input("Enter city: ")

response=requests.get(f"https://api.openweathermap.org/data/2.5/
weather?q={city}&APPID={api_key}&units=Metric")

a=response.json()

if 'message' in a:

    print("City not Found!")

else:

    print("\nCity:",city)
    print("Temperature:",a['main']['temp'],"C")

    print("Humidity:",a['main']['humidity'])

    print("Sunrise(IST):",datetime.datetime.fromtimestamp(a['sys']['sunrise']))

    print(“Sunset(IST):”,datetime.datetime.fromtimestamp(a['sys']['sunset']))
```

Output: (screenshot)

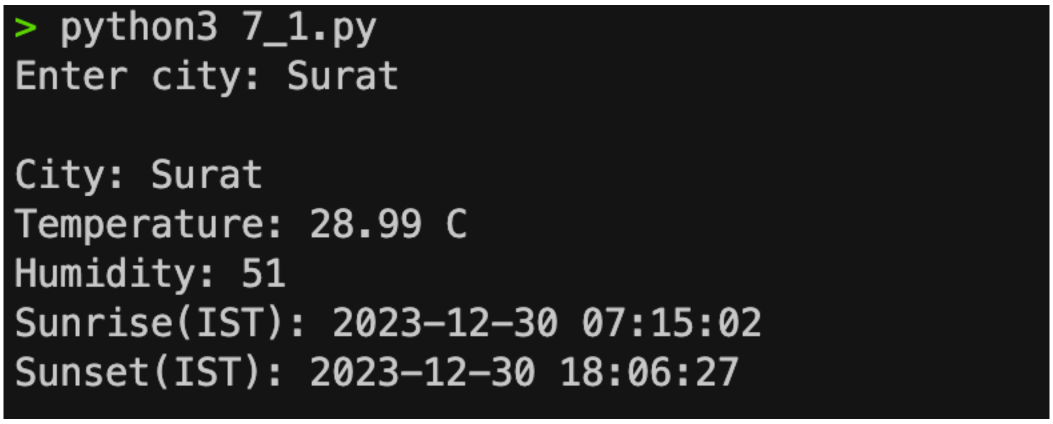


```
> python3 7_1.py
Enter city: Mumbai

City: Mumbai
Temperature: 31.99 C
Humidity: 48
Sunrise(IST): 2023-12-30 07:10:48
Sunset(IST): 2023-12-30 18:10:34
```

Test Case: Any two (screenshot)



```
Enter city: Navi Mumbai

City: Navi Mumbai
Temperature: 31.97 C
Humidity: 48
Sunrise(IST): 2023-12-30 07:10:10
Sunset(IST): 2023-12-30 18:09:51
```



```
> python3 7_1.py
Enter city: Surat

City: Surat
Temperature: 28.99 C
Humidity: 51
Sunrise(IST): 2023-12-30 07:15:02
Sunset(IST): 2023-12-30 18:06:27
```

Conclusion : The code effectively showcases how to interact with web APIs in Python to retrieve and process external data. It highlights the use of requests for HTTP communication, JSON parsing, API key management, error handling, and date formatting.It provides a foundation for building applications that leverage real-time data from various online sources.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 7.1

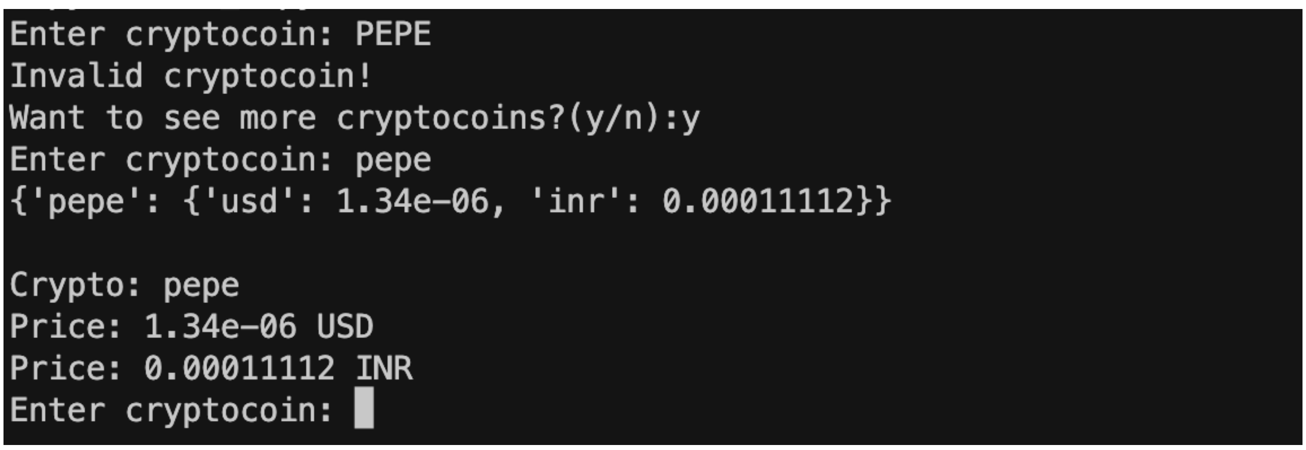
Title : Write a program to use the ‘API’ of crypto currency**.**

Theory: Cryptocurrency APIs provide a way for developers to access real-time data about cryptocurrencies, including prices, market data, and other relevant information.

Code:

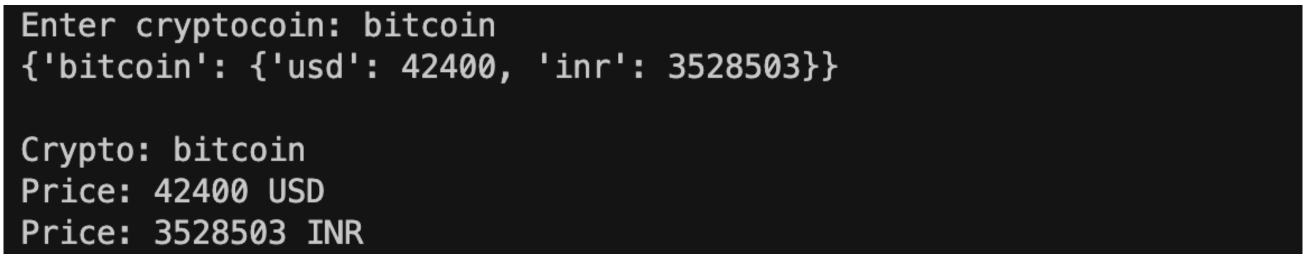
```
API_KEY='CG-Kjmr47XUisC8wTQ75jsf7wAS'

import requests

while True:

 coin=input("Enter cryptocoin: ")

 response = requests.get(f"https://api.coingecko.com/api/v3/simple/price?
ids={coin}&vs_currencies=usd,inr&x_cg_demo_api_key={API_KEY}")



 a=response.json()



 if coin in a:

     print(a)

     print("\nCrypto:",coin)

     print("Price:",a[coin]['usd'],"USD")

     print("Price:",a[coin]['inr'],"INR")



 else:
    print("Invalid cryptocoin!")

    b=input("Want to see more cryptocoins?(y/n):")

 if b.lower() == "n":

    break
```

Output:



```
Enter cryptocoin: PEPE
Invalid cryptocoin!
Want to see more cryptocoins?(y/n):y
Enter cryptocoin: pepe
{'pepe': {'usd': 1.34e-06, 'inr': 0.00011112}}

Crypto: pepe
Price: 1.34e-06 USD
Price: 0.00011112 INR
Enter cryptocoin: █
```

Test Case:



```
Enter cryptocoin: bitcoin
{'bitcoin': {'usd': 42400, 'inr': 3528503}}

Crypto: bitcoin
Price: 42400 USD
Price: 3528503 INR
```

Conclusion: The program demonstrates how to use a cryptocurrency API (CoinGecko) to retrieve and display the current price of a specified cryptocurrency. APIs offer a powerful means for developers to incorporate dynamic data into applications, enhancing functionality and keeping information up-to-date.