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

PYTHON PROGRAMMING LAB

Prepared by:

Name of Student: Manmeet Singh

Roll No:16

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Exp. No	List of Experiment
1	1.1 Write a program to compute Simple Interest.

	1.2 Write a program to perform arithmetic, Relational operators.
	1.3 Write a program to find whether a given no is even & odd.
	1.4 Write a program to print first n natural number & their sum.
	1.5 Write a program to determine whether the character entered is a Vowel or not .
	1.6 Write a program to find whether given number is an Armstrong Number.
	1.7 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 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.

	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: Manmeet Singh

Roll Number: 16

Experiment No: 1.1

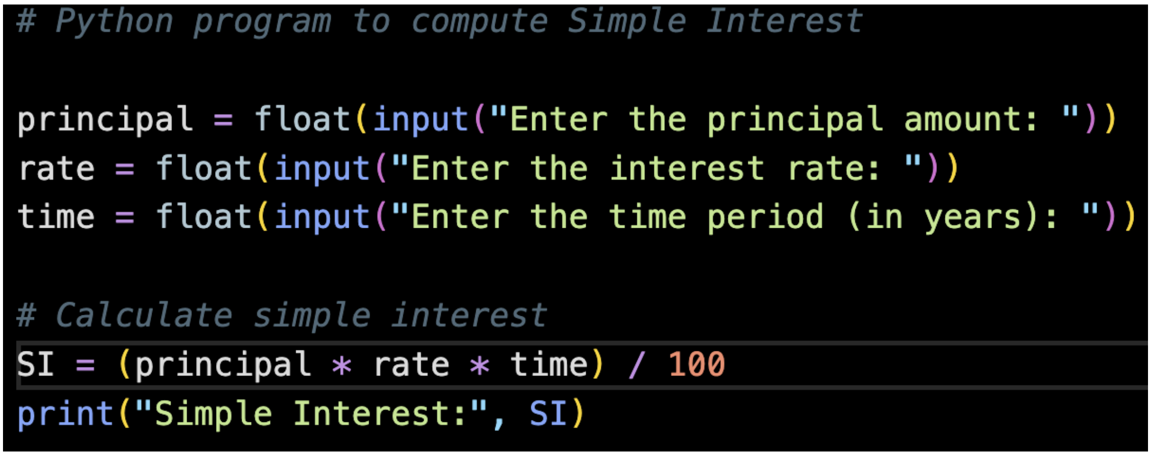
Title: Write a program to compute Simple Interest.

Theory:

- Gathers essential data: The program prompts the user to provide the principal amount, interest rate, and time period.

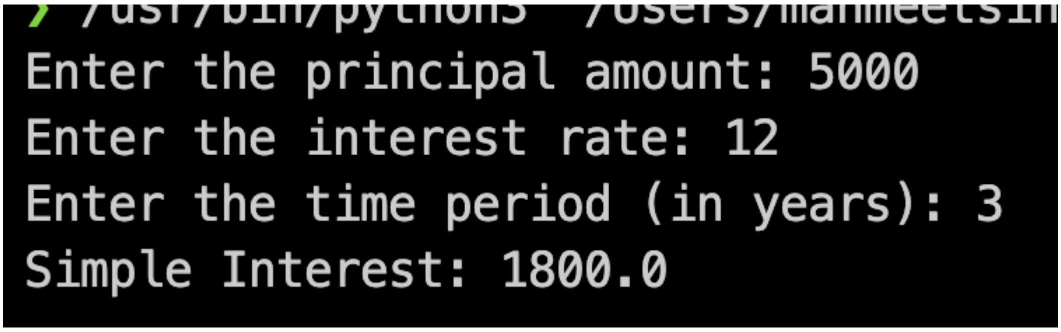
- Converts input to numbers: It transforms the input values into floating-point numbers for precise calculations.
- Applies the formula: It directly implements the Simple Interest formula: SI = (P * R * T) / 100.
- Calculates and displays: It determines the simple interest using the formula and presents the result to the user.

Code:



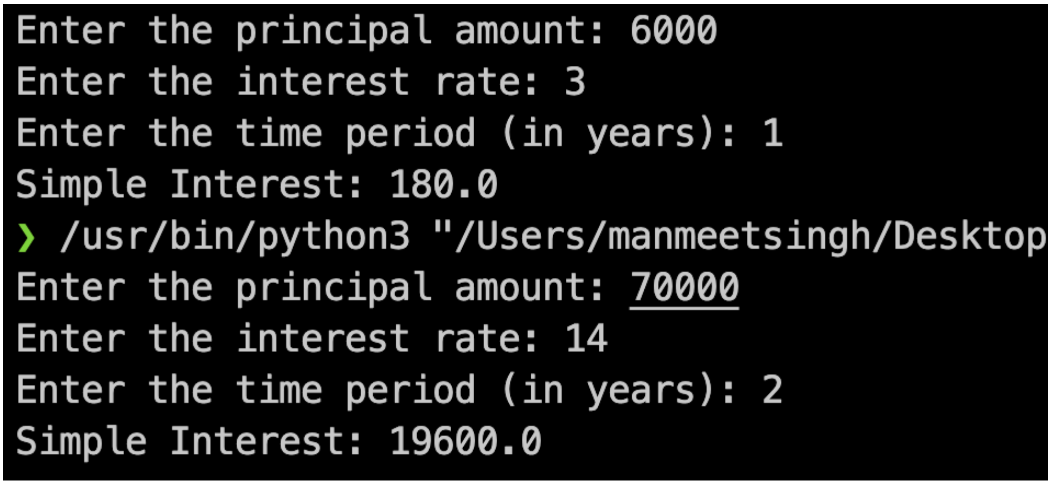
```python
# Python program to compute Simple Interest

principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the interest rate: "))
time = float(input("Enter the time period (in years): "))

# Calculate simple interest
SI = (principal * rate * time) / 100
print("Simple Interest:", SI)
```

Output: (screenshot)



```
❯ /usr/bin/python3 /Users/manmeetsin
Enter the principal amount: 5000
Enter the interest rate: 12
Enter the time period (in years): 3
Simple Interest: 1800.0
```

Test Case: Any two (screenshot)



```
Enter the principal amount: 6000
Enter the interest rate: 3
Enter the time period (in years): 1
Simple Interest: 180.0
❯ /usr/bin/python3 "/Users/manmeetsingh/Desktop
Enter the principal amount: 70000
Enter the interest rate: 14
Enter the time period (in years): 2
Simple Interest: 19600.0
```

Conclusion:

- **Simply calculates interest: This code effectively calculates simple interest for basic financial scenarios. The Program accurately determines the simple interest based on user input :- principal, rate and time.**

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 1.2

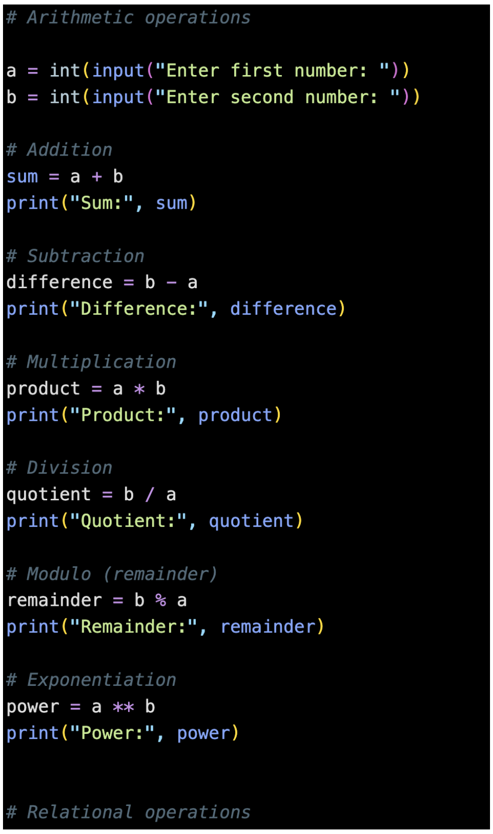
Title: Write a program to perform arithmetic, Relational operations

Theory:

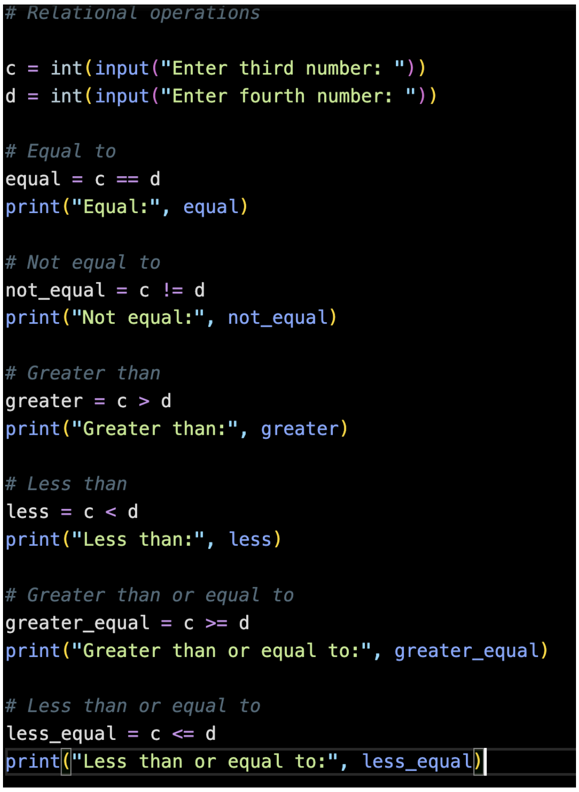
- Arithmetic Operations:The program performs basic calculations using various arithmetic operators, showcasing their syntax and functionality.

- Relational Operations: It demonstrates how to compare values using relational operators, producing boolean (True/False) results.

Code

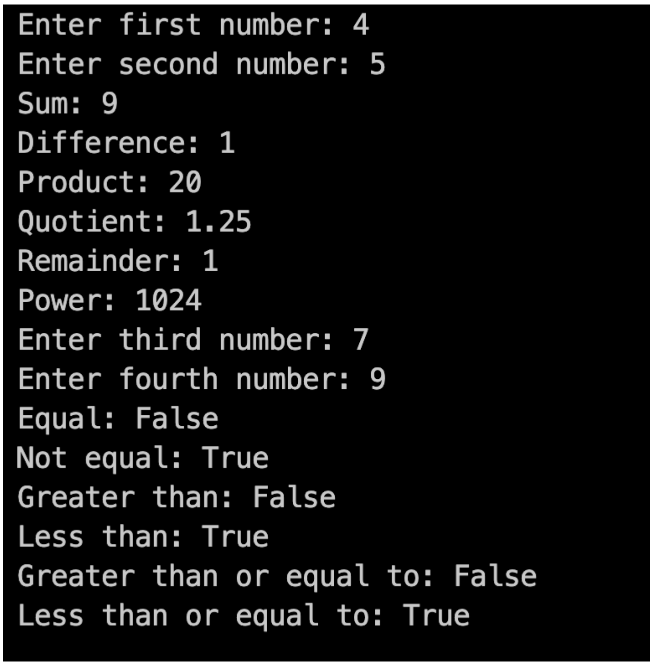


```
# Arithmetic operations

a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

# Addition
sum = a + b
print("Sum:", sum)

# Subtraction
difference = b - a
print("Difference:", difference)

# Multiplication
product = a * b
print("Product:", product)

# Division
quotient = b / a
print("Quotient:", quotient)

# Modulo (remainder)
remainder = b % a
print("Remainder:", remainder)

# Exponentiation
power = a ** b
print("Power:", power)


# Relational operations
```



```python
# Relational operations

c = int(input("Enter third number: "))
d = int(input("Enter fourth number: "))

# Equal to
equal = c == d
print("Equal:", equal)

# Not equal to
not_equal = c != d
print("Not equal:", not_equal)

# Greater than
greater = c > d
print("Greater than:", greater)

# Less than
less = c < d
print("Less than:", less)

# Greater than or equal to
greater_equal = c >= d
print("Greater than or equal to:", greater_equal)

# Less than or equal to
less_equal = c <= d
print("Less than or equal to:", less_equal)
```

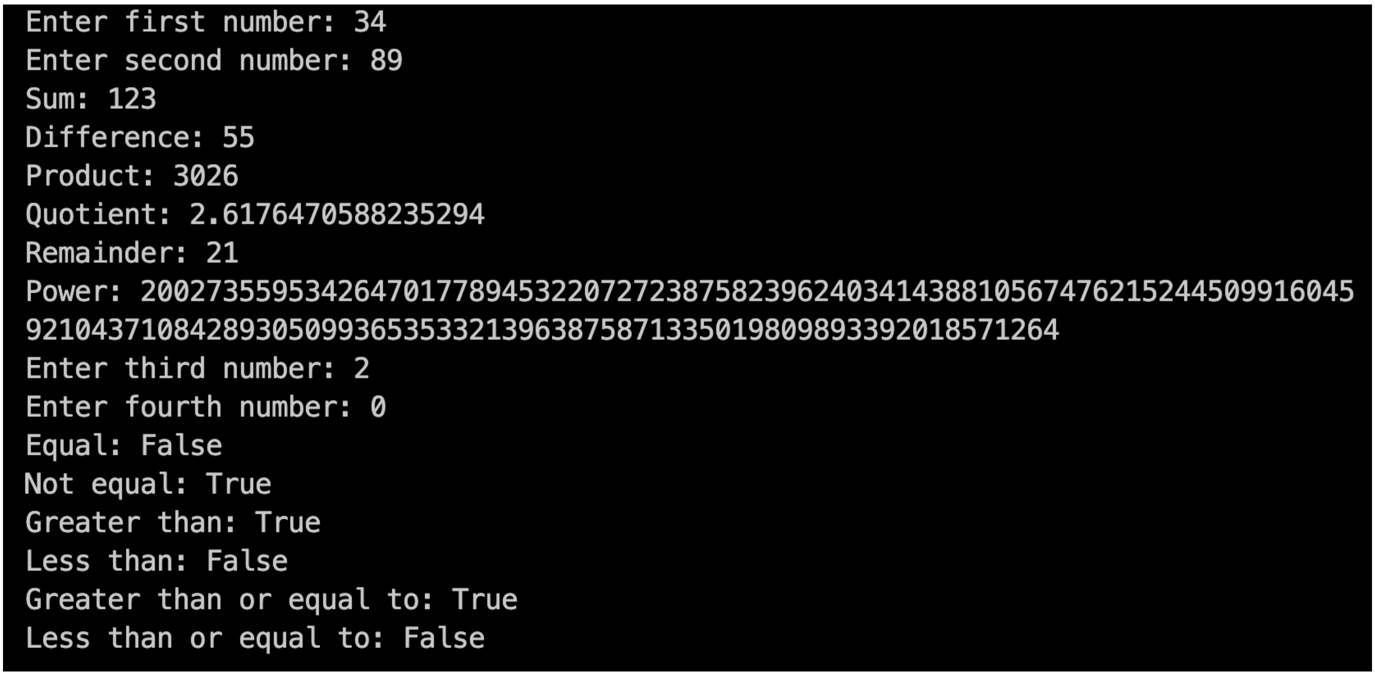
Output:



```
Enter first number: 4
Enter second number: 5
Sum: 9
Difference: 1
Product: 20
Quotient: 1.25
Remainder: 1
Power: 1024
Enter third number: 7
Enter fourth number: 9
Equal: False
Not equal: True
Greater than: False
Less than: True
Greater than or equal to: False
Less than or equal to: True
```

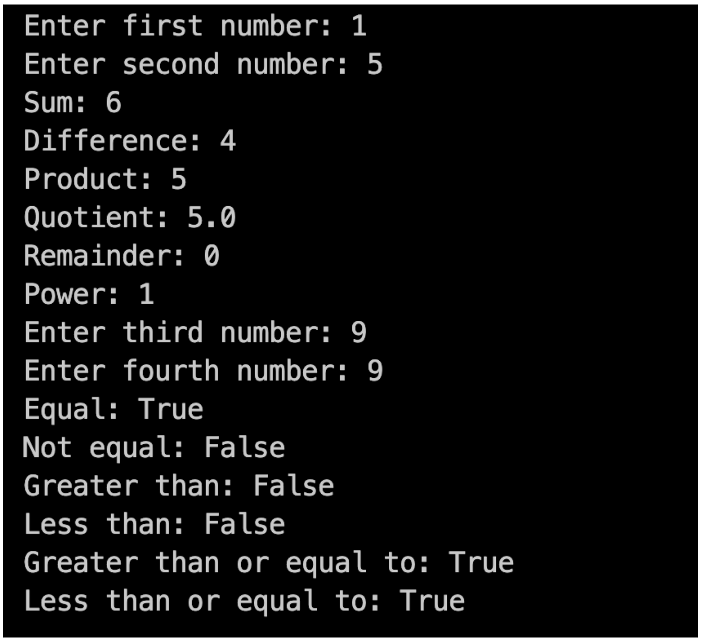
Test Cases:

1)



```
Enter first number: 34
Enter second number: 89
Sum: 123
Difference: 55
Product: 3026
Quotient: 2.6176470588235294
Remainder: 21
Power: 200273559534264701778945322072723875823962403414388105674762152445099160459
21043710842893050993653533213963875871335019809893392018571264
Enter third number: 2
Enter fourth number: 0
Equal: False
Not equal: True
Greater than: True
Less than: False
Greater than or equal to: True
Less than or equal to: False
```

2)



```
Enter first number: 1
Enter second number: 5
Sum: 6
Difference: 4
Product: 5
Quotient: 5.0
Remainder: 0
Power: 1
Enter third number: 9
Enter fourth number: 9
Equal: True
Not equal: False
Greater than: False
Less than: False
Greater than or equal to: True
Less than or equal to: True
```

Conclusion:

The code effectively illustrates both arithmetic and relational operations in Python.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 1.3

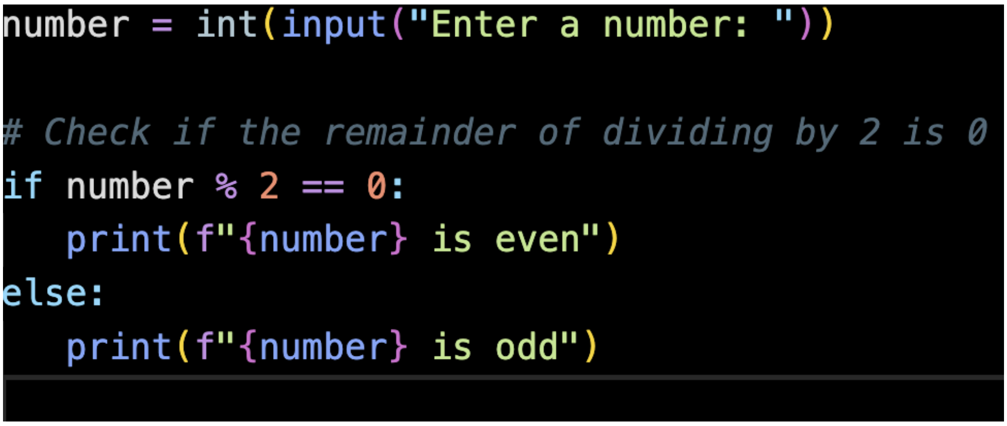
Title: Write a program to identify whether a no is even or odd

Theory:

- Key Concept: The code determines evenness or oddness of a number based on its divisibility by 2.

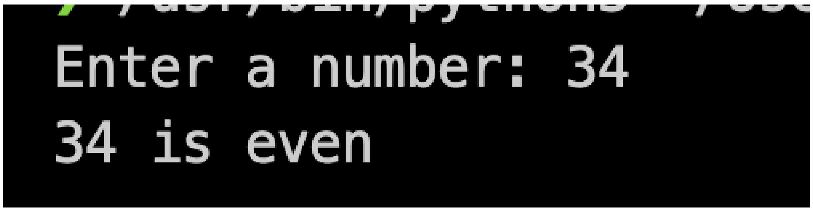
- Conditional Statement: An if statement checks the remainder:
    - If the remainder is 0, the number is even.
    - Otherwise, it's odd.

Code



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

# Check if the remainder of dividing by 2 is 0
if number % 2 == 0:
   print(f"{number} is even")
else:
   print(f"{number} is odd")
```

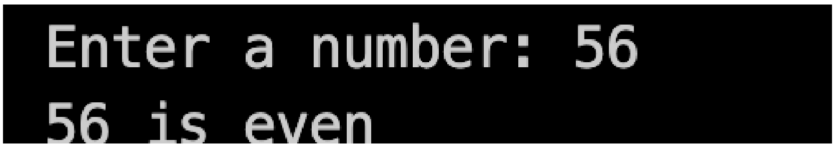
Output



```
Enter a number: 34
34 is even
```

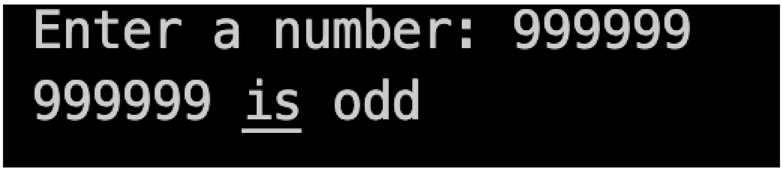
Test Cases

1)



```
Enter a number: 56
56 is even
```

2)



```
Enter a number: 999999
999999 is odd
```

Conclusion:

This code offers a straightforward and efficient way to determine whether a number is even or odd.

Name of Student: Manmeet Singh

Roll Number: 16

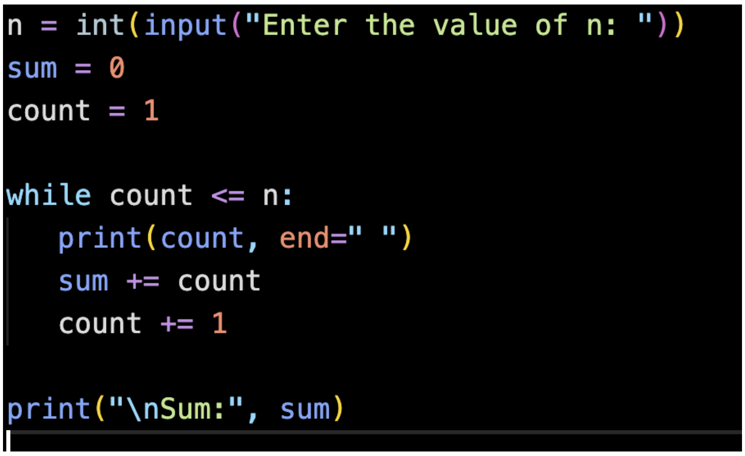
Experiment No: 1.4

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

Theory:

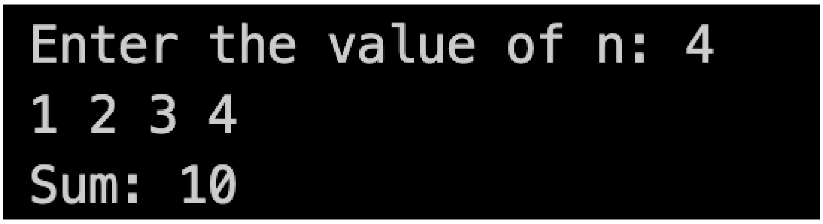
It starts by defining n, representing the desired number of natural numbers to be processed. Natural numbers are positive numbers beginning from 1.

Code



```
n = int(input("Enter the value of n: "))
sum = 0
count = 1

while count <= n:
    print(count, end=" ")
    sum += count
    count += 1

print("\nSum:", sum)
```

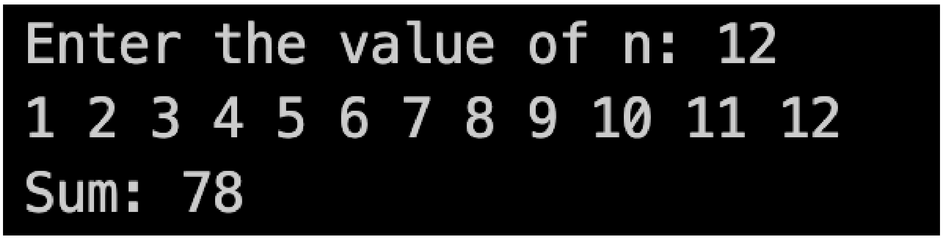
Output



```
Enter the value of n: 4
1 2 3 4
Sum: 10
```

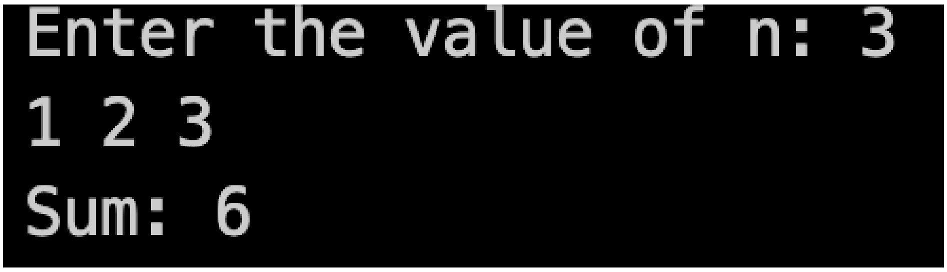
Test Cases

1)



```
Enter the value of n: 12
1 2 3 4 5 6 7 8 9 10 11 12
Sum: 78
```

2)



```
Enter the value of n: 3
1 2 3
Sum: 6
```

Conclusion: This code effectively generates and prints the first `n` natural numbers while simultaneously calculating their sum.

Name of Student: Manmeet Singh

Roll Number: 16

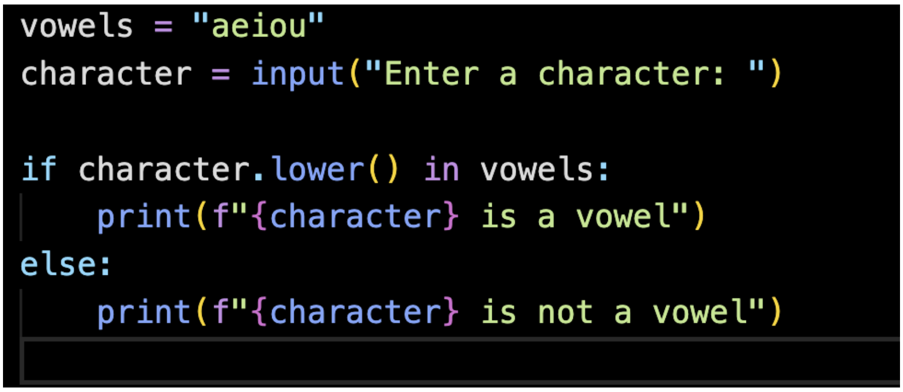
Experiment No: 1.5

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

Theory:

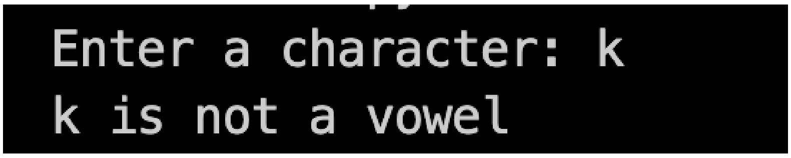
Vowels are the open, core sounds of syllables (a, e, i, o, u). Consonants shape those sounds and add detail, like building blocks around vowels (b, c, d, f...).

Code



```python
vowels = "aeiou"
character = input("Enter a character: ")

if character.lower() in vowels:
    print(f"{character} is a vowel")
else:
    print(f"{character} is not a vowel")
```

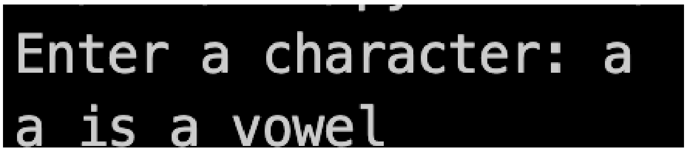
Output



```
Enter a character: k
k is not a vowel
```

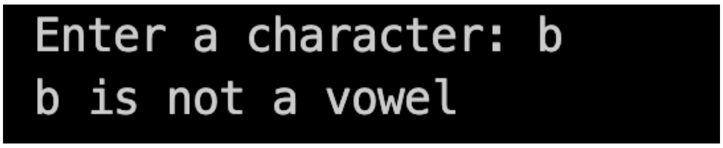
Test Cases

1)



```
Enter a character: a
a is a vowel
```

2)



```
Enter a character: b
b is not a vowel
```

Conclusion

The code offers a straightforward and efficient approach to determining whether a character is a vowel or not.

Name of Student: Manmeet Singh

Roll Number: 16

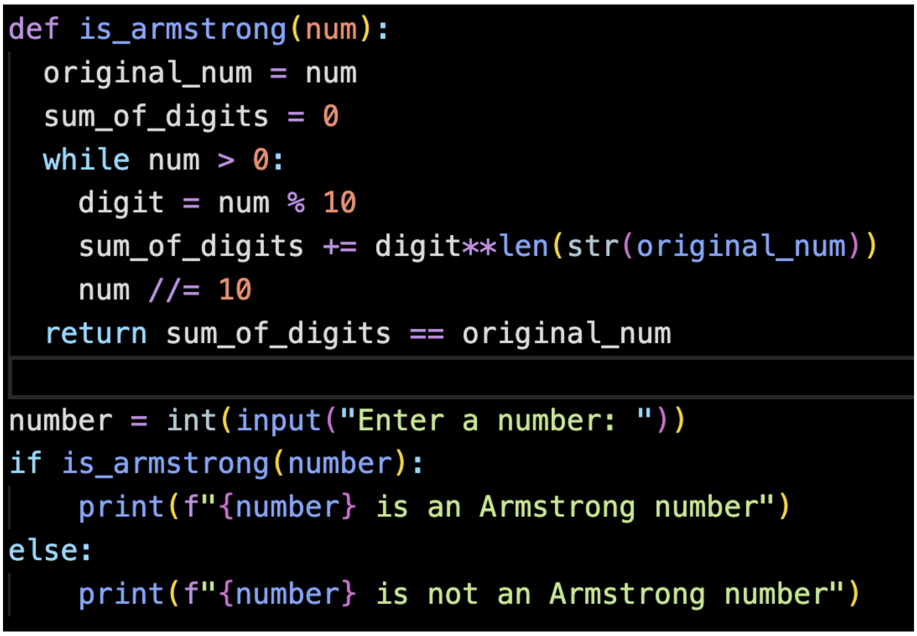
Experiment No: 1.6

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

Theory:

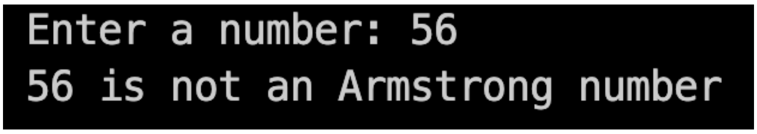
Armstrong numbers are special digits like 153 or 371. Each digit raised to the number of digits, when summed, equals the original number.

Code



```python
def is_armstrong(num):
  original_num = num
  sum_of_digits = 0
  while num > 0:
    digit = num % 10
    sum_of_digits += digit**len(str(original_num))
    num //= 10
  return sum_of_digits == original_num

number = int(input("Enter a number: "))
if is_armstrong(number):
    print(f"{number} is an Armstrong number")
else:
    print(f"{number} is not an Armstrong number")
```

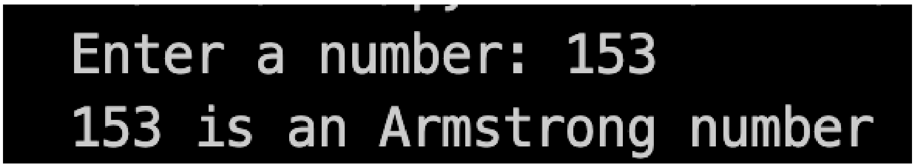
Output



```
Enter a number: 56
56 is not an Armstrong number
```

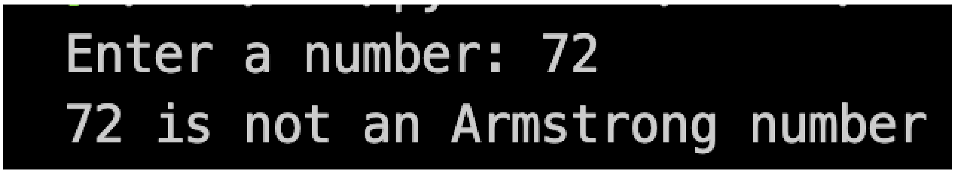
Test Cases

1)



```
Enter a number: 153
153 is an Armstrong number
```

2)



```
Enter a number: 72
72 is not an Armstrong number
```

Conclusion

The code effectively determines whether a given number is an Armstrong number, relying on a well-defined function.

Name of Student: Manmeet Singh

Roll Number: 16

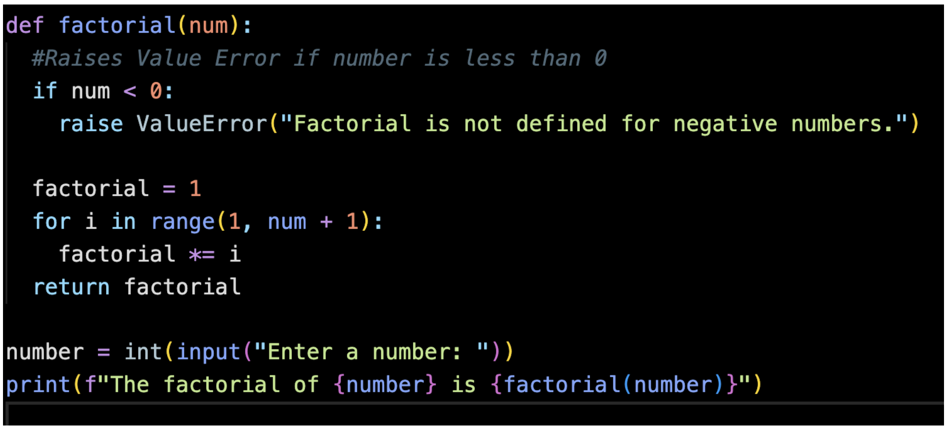
Experiment No: 1.7

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

Theory:

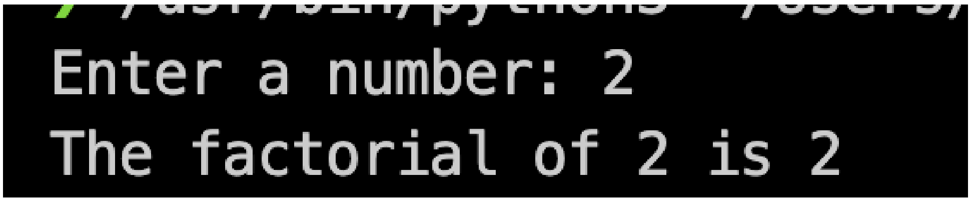
Factorial is a number's rapid descent down a multiplication slide! It's the product of all positive integers from 1 up to that number. Example: 5! (5 factorial) equals 1*2345* = 120.

Code



```python
def factorial(num):
  #Raises Value Error if number is less than 0
  if num < 0:
    raise ValueError("Factorial is not defined for negative numbers.")

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

number = int(input("Enter a number: "))
print(f"The factorial of {number} is {factorial(number)}")
```

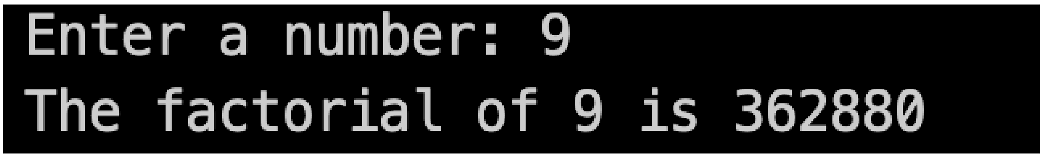
Output



```
Enter a number: 2
The factorial of 2 is 2
```

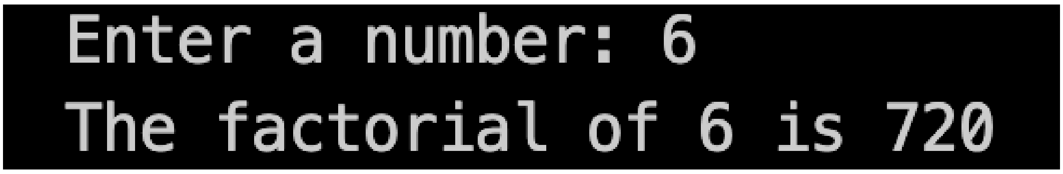
Test Cases

1)



```
Enter a number: 9
The factorial of 9 is 362880
```

2)



```
Enter a number: 6
The factorial of 6 is 720
```

Conclusion

It serves as a clear example of functions, loops, mathematical operations, and error handling in programming.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 1.8 i)

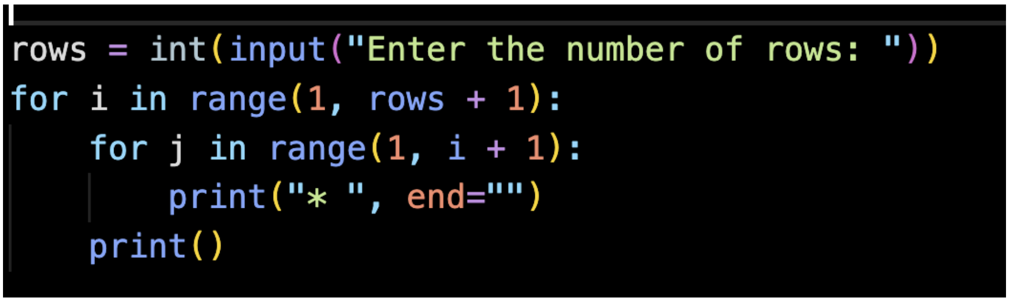
Title: Write a program to print the following pattern

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

Theory:

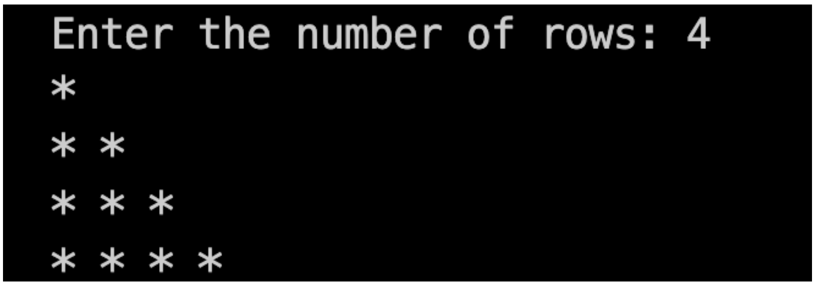
The code effectively generates a triangular pattern of stars using nested for loops. The outer loop iterates through the desired number of rows, while the inner loop dynamically prints stars within each row, increasing in number from one to the current row index. This demonstrates the ability of nested loops to create structured patterns with varying elements.

Code:



```python
rows = int(input("Enter the number of rows: "))
for i in range(1, rows + 1):
    for j in range(1, i + 1):
        print("* ", end="")
    print()
```

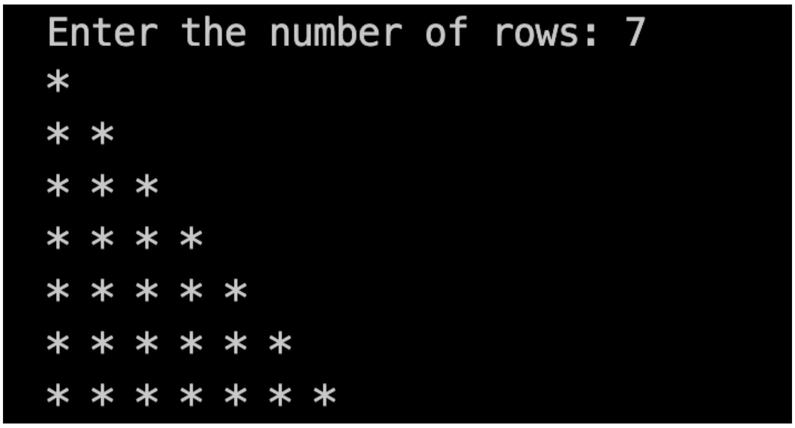
Output



```
Enter the number of rows: 4
*
* *
* * *
* * * *
```

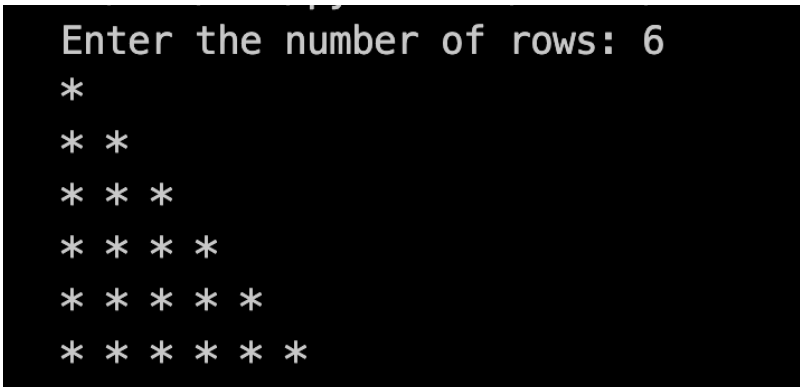
Test Cases

1)



```
Enter the number of rows: 7
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
```

2)



```
Enter the number of rows: 6
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

Conclusion: Nested loops enable managing multiple levels of iteration for tasks involving structured patterns.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 1.8 ii)

Title: Write a program to print following pattern

ii)

1

2 2

3 3 3

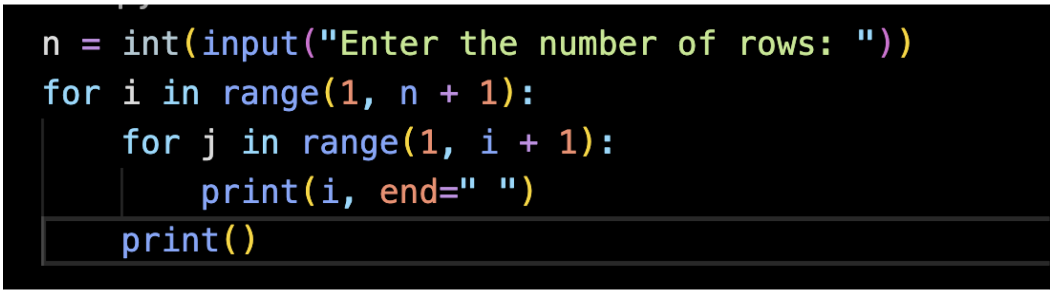
4 4 4 4

5 5 5 5 5

Theory:

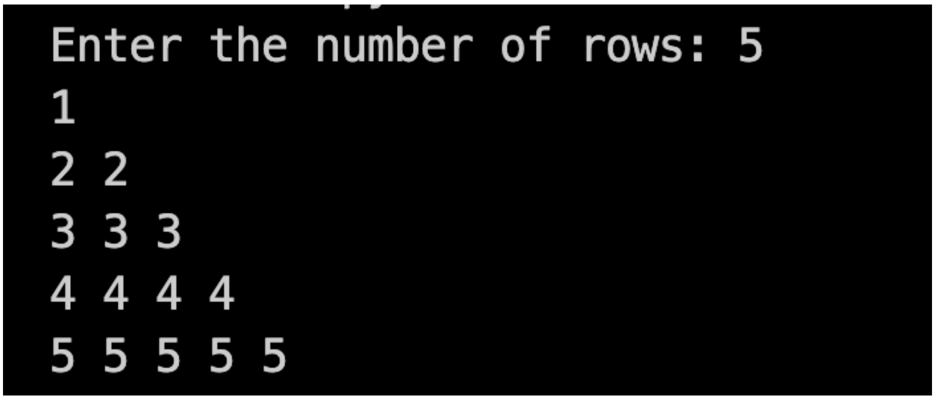
The code utilizes nested loops to generate a triangular pattern of integers. The outer loop defines the number of rows, while the inner loop iterates within each row, progressively printing the current row number, creating a structured arrangement of ascending numbers.

Code



```python
n = int(input("Enter the number of rows: "))
for i in range(1, n + 1):
    for j in range(1, i + 1):
        print(i, end=" ")
    print()
```

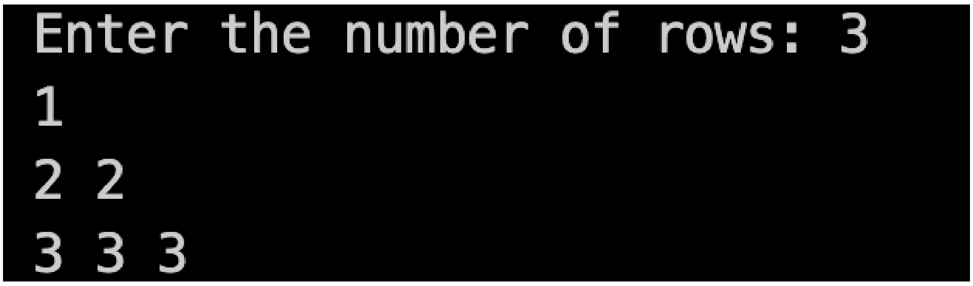
Output



```
Enter the number of rows: 5
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

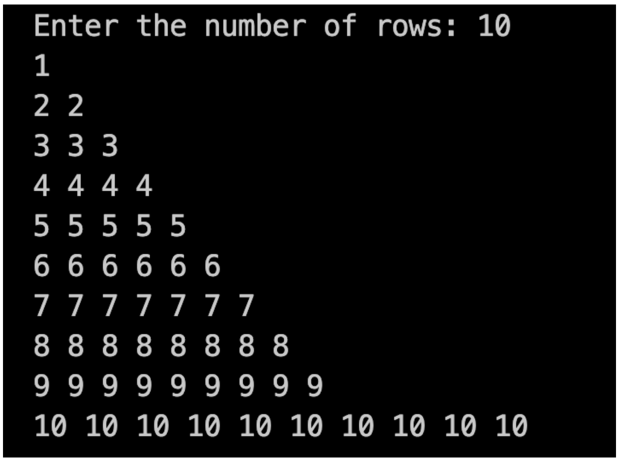
Test Cases

1)



```
Enter the number of rows: 3
1
2 2
3 3 3
```

2)



```
Enter the number of rows: 10
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
10 10 10 10 10 10 10 10 10 10
```

Conclusion: Nested loops effectively construct a triangular pattern of ascending integers, demonstrating their ability to create structured output with varying elements.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 1.8 iii)

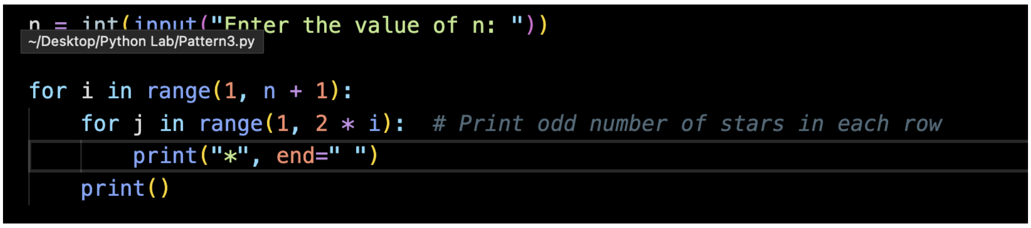
Title: Write a program to print the following pattern

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

Theory:

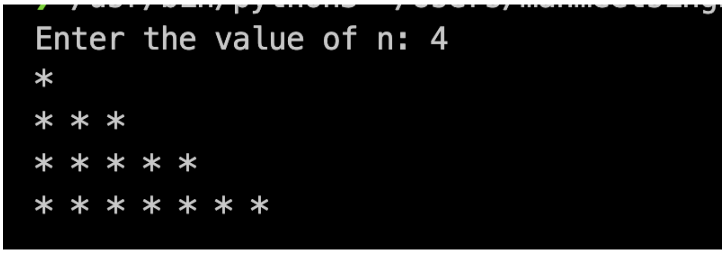
The code utilizes nested loops to construct a diamond-shaped pattern of stars. The outer loop determines the number of rows, while the inner loop dynamically prints stars based on twice the current row number, ensuring an increasing pattern.

Code



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

for i in range(1, n + 1):
    for j in range(1, 2 * i):  # Print odd number of stars in each row
        print("*", end=" ")
    print()
```

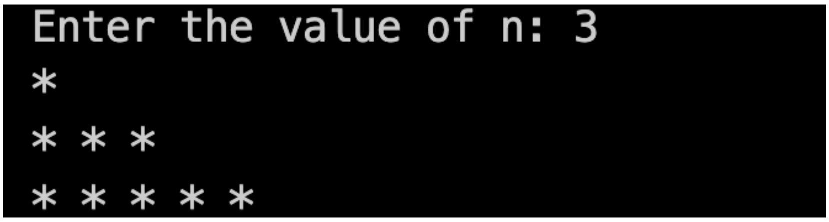
Output



```
Enter the value of n: 4
*
* * *
* * * * *
* * * * * * *
```

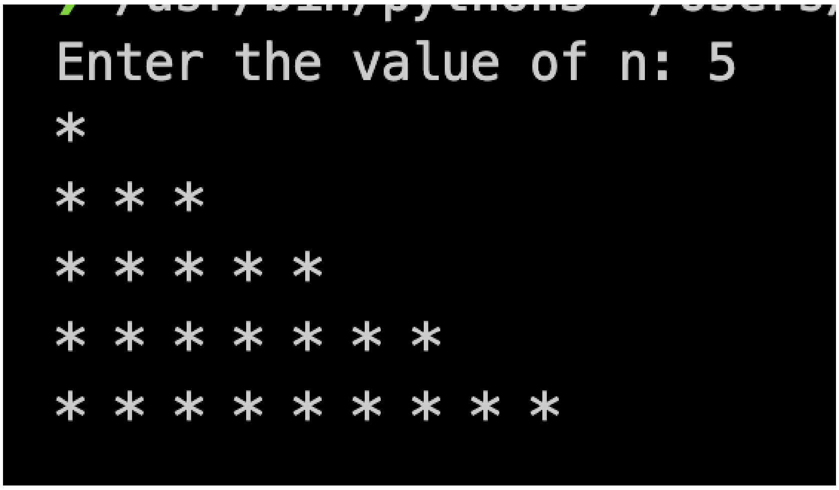
Test Cases

1)



```
Enter the value of n: 3
*
* * *
* * * * *
```

2)



```
Enter the value of n: 5
*
* * *
* * * * *
* * * * * * *
* * * * * * * * *
```

Conclusion: Nested loops effectively create a diamond-shaped pattern of stars, demonstrating their ability to generate structured visual output with dynamic elements.

Name of Student: Manmeet Singh

Roll Number: 16

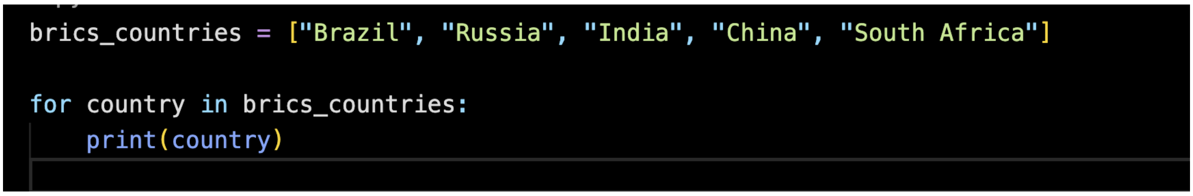
Experiment No: 2.1

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

Theory:

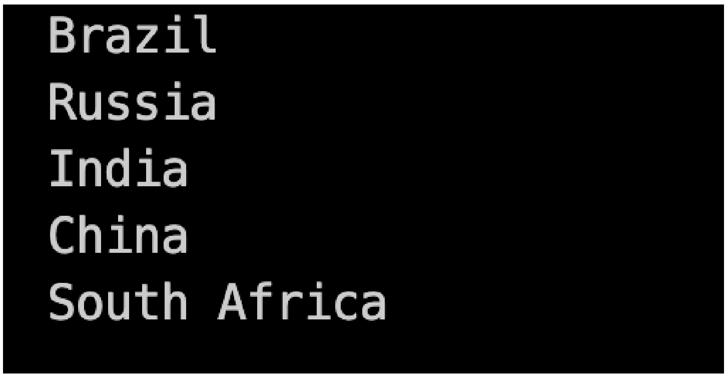
The code leverages a for loop to iterate through the list, showcasing sequential access and element retrieval for structured data processing.

Code



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

for country in brics_countries:
    print(country)
```

Output



```
Brazil
Russia
India
China
South Africa
```

Conclusion: The `for` loop effectively prints each country in the list, demonstrating its ability to traverse and process elements within structured data collections.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 2.2

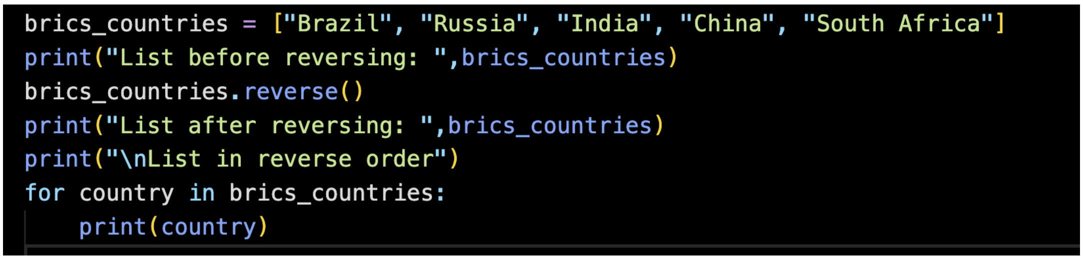
Title: Write a program to traverse a list in reverse order.

1. **using Reverse method.**

Theory:

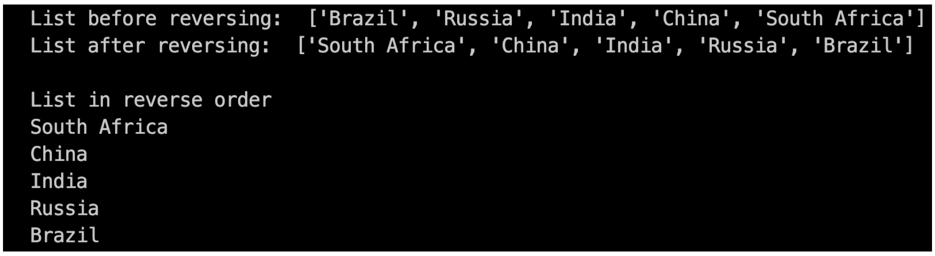
The code employs the reverse() method to invert the list's order in-place, enabling subsequent traversal in reverse sequence for flexible data manipulation.

Code



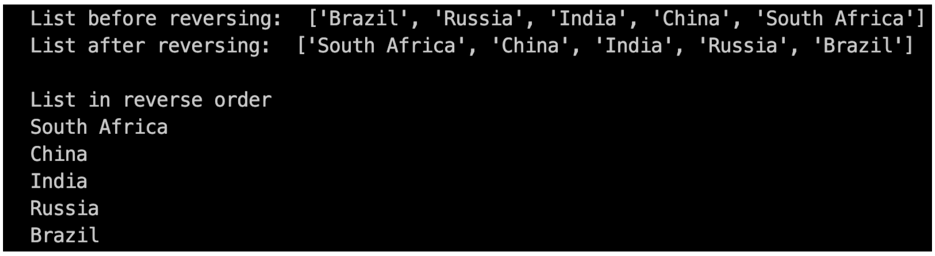
```python
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]
print("List before reversing: ",brics_countries)
brics_countries.reverse()
print("List after reversing: ",brics_countries)
print("\nList in reverse order")
for country in brics_countries:
    print(country)
```

Output



```
List before reversing:  ['Brazil', 'Russia', 'India', 'China', 'South Africa']
List after reversing:  ['South Africa', 'China', 'India', 'Russia', 'Brazil']

List in reverse order
South Africa
China
India
Russia
Brazil
```

Test Case



```
List before reversing:  ['Brazil', 'Russia', 'India', 'China', 'South Africa']
List after reversing:  ['South Africa', 'China', 'India', 'Russia', 'Brazil']

List in reverse order
South Africa
China
India
Russia
Brazil
```

Conclusion: The reverse() method effectively inverts the list's order, empowering subsequent traversal and manipulation in a reversed sequence.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 2.2

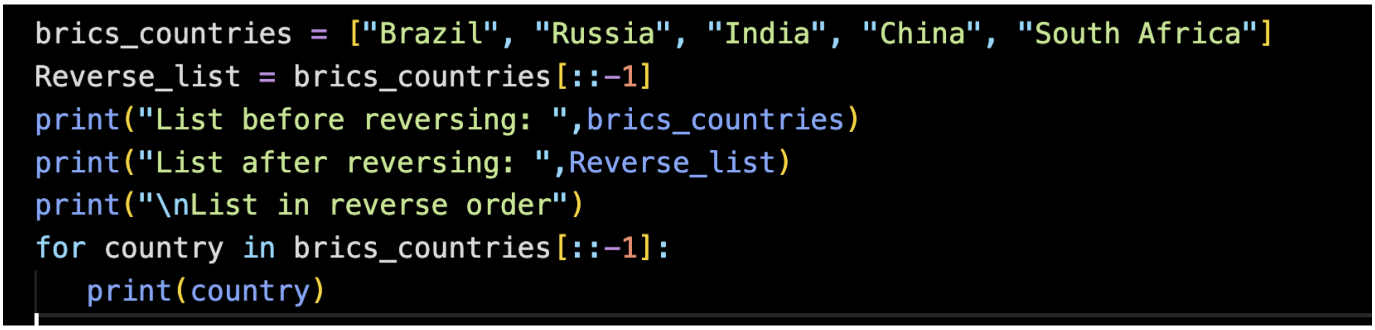
Title: Write a program to traverse a list in reverse order.

2. By using slicing

Theory:

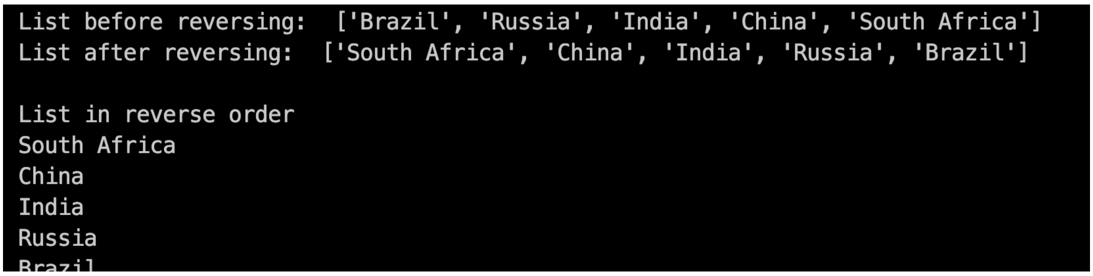
The code utilizes list slicing with a step of -1 to generate a reversed view of the list, enabling traversal in reverse order without modifying the original data structure

Code



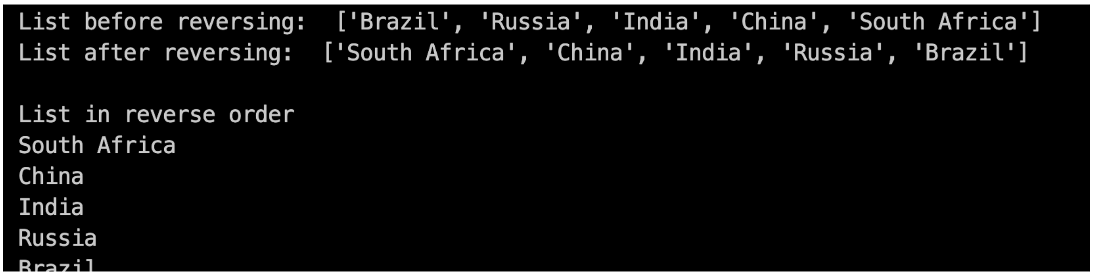
```python
brics_countries = ["Brazil", "Russia", "India", "China", "South Africa"]
Reverse_list = brics_countries[::-1]
print("List before reversing: ",brics_countries)
print("List after reversing: ",Reverse_list)
print("\nList in reverse order")
for country in brics_countries[::-1]:
    print(country)
```

Output



```
List before reversing:  ['Brazil', 'Russia', 'India', 'China', 'South Africa']
List after reversing:  ['South Africa', 'China', 'India', 'Russia', 'Brazil']

List in reverse order
South Africa
China
India
Russia
Brazil
```

Test Case



```
List before reversing:  ['Brazil', 'Russia', 'India', 'China', 'South Africa']
List after reversing:  ['South Africa', 'China', 'India', 'Russia', 'Brazil']

List in reverse order
South Africa
China
India
Russia
Brazil
```

Conclusion: List slicing with a negative step effectively creates a reversed view for traversal, offering a non-destructive approach to reverse-order processing.

Name of Student: Manmeet Singh

Roll Number: 16

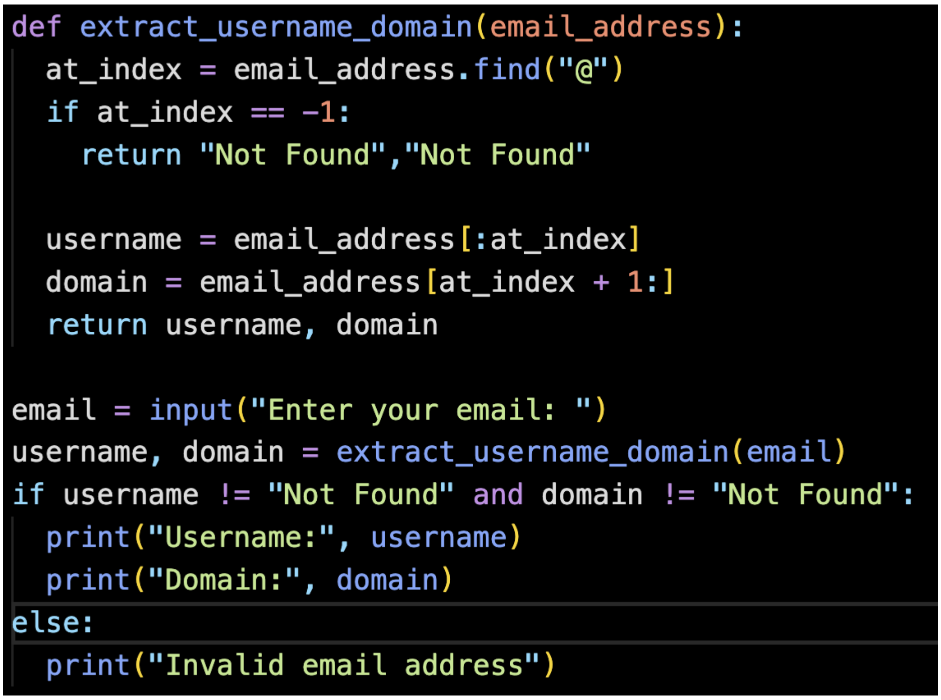
Experiment No: 2.3

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

Theory:

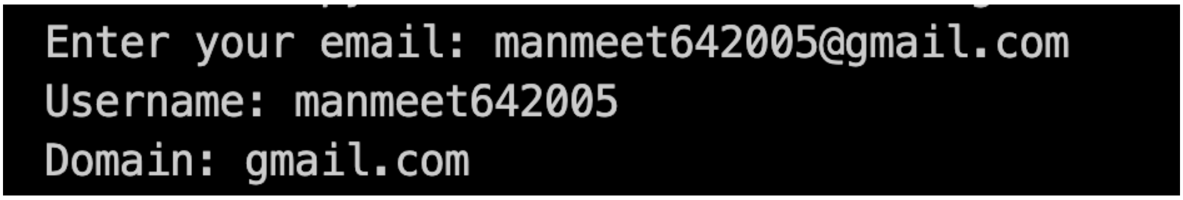
The code takes an email address and separates it into two parts: the name before the "@" (username) and the part after (domain). It works like a mail sorter, putting each part in its own box (a tuple) so you can see them clearly. If the email isn't even a valid address, it throws it away instead.

Code



```python
def extract_username_domain(email_address):
  at_index = email_address.find("@")
  if at_index == -1:
    return "Not Found","Not Found"

  username = email_address[:at_index]
  domain = email_address[at_index + 1:]
  return username, domain

email = input("Enter your email: ")
username, domain = extract_username_domain(email)
if username != "Not Found" and domain != "Not Found":
  print("Username:", username)
  print("Domain:", domain)
else:
  print("Invalid email address")
```

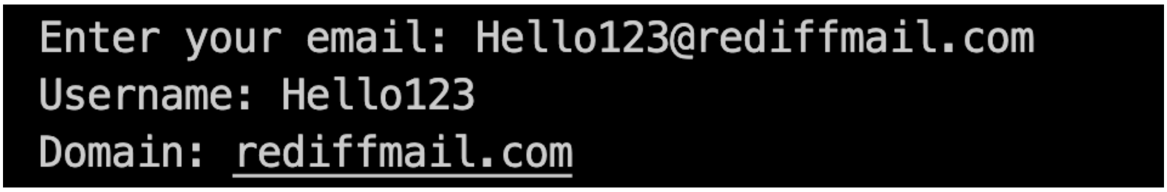
Output



```
Enter your email: manmeet642005@gmail.com
Username: manmeet642005
Domain: gmail.com
```

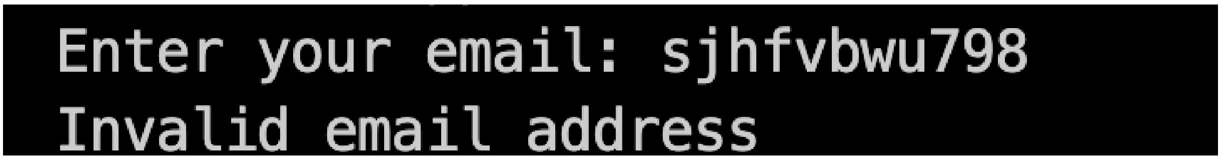
Test Case

1)



```
Enter your email: Hello123@rediffmail.com
Username: Hello123
Domain: rediffmail.com
```

2)



```
Enter your email: sjhfvbwu798
Invalid email address
```

Conclusion: The code effectively extracts and organizes email components into a tuple, demonstrating efficient data extraction and structural organization for email processing tasks.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 2.4

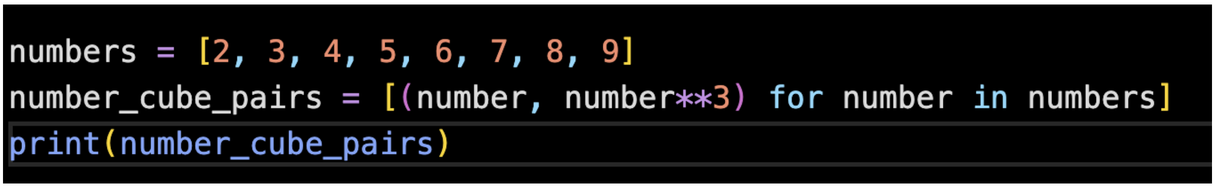
Title: Write a program to create a list of tuples from given list having number and add its cube in tuple.

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

Theory:

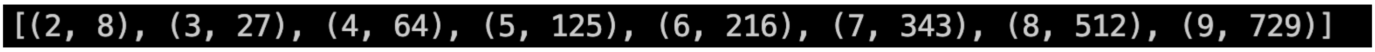
The code leverages list comprehension to efficiently create tuples pairing each number with its cube, demonstrating concise data transformation and structured output generation.

Code



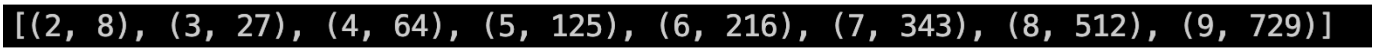
```
numbers = [2, 3, 4, 5, 6, 7, 8, 9]
number_cube_pairs = [(number, number**3) for number in numbers]
print(number_cube_pairs)
```

Output



```
[(2, 8), (3, 27), (4, 64), (5, 125), (6, 216), (7, 343), (8, 512), (9, 729)]
```

Test Case



```
[(2, 8), (3, 27), (4, 64), (5, 125), (6, 216), (7, 343), (8, 512), (9, 729)]
```

Conclusion:

The code generates pairs of numbers and their respective cubes in a list of tuples.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 2.5

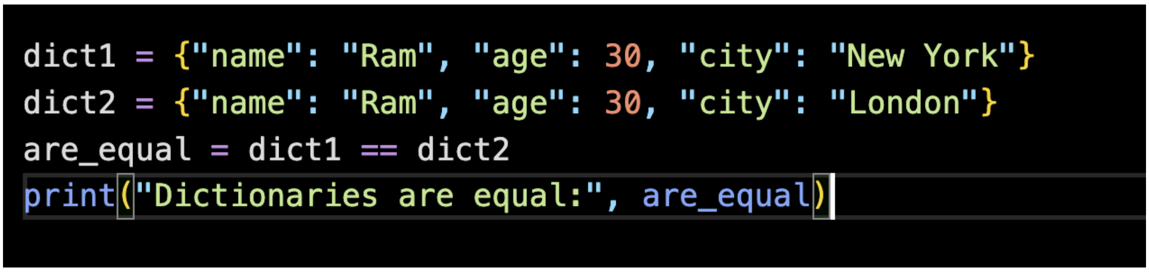
Title: Write a program to compare two dictionaries in Python?

(By using == operator)

Theory:

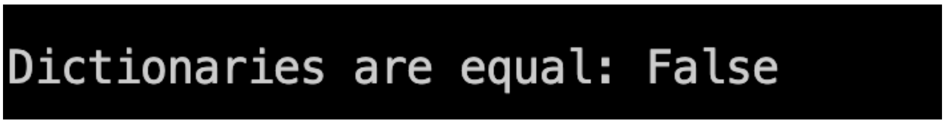
The code compares two dictionaries (dict1 and dict2) by checking if their key-value pairs match, determining their equality, and displaying the result.

Code



```
dict1 = {"name": "Ram", "age": 30, "city": "New York"}
dict2 = {"name": "Ram", "age": 30, "city": "London"}
are_equal = dict1 == dict2
print("Dictionaries are equal:", are_equal)
```

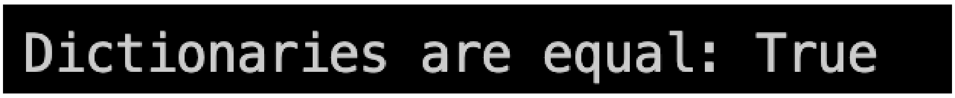
Output



```
Dictionaries are equal: False
```

Test Case

1)



```
Dictionaries are equal: True
```

Conclusion: The code compares two dictionaries, dict1 and dict2, checking if their key-value pairs are identical and displays whether they are equal or not. In this case, they are considered unequal due to the differing city values.

Name of Student: Manmeet Singh

Roll Number: 16

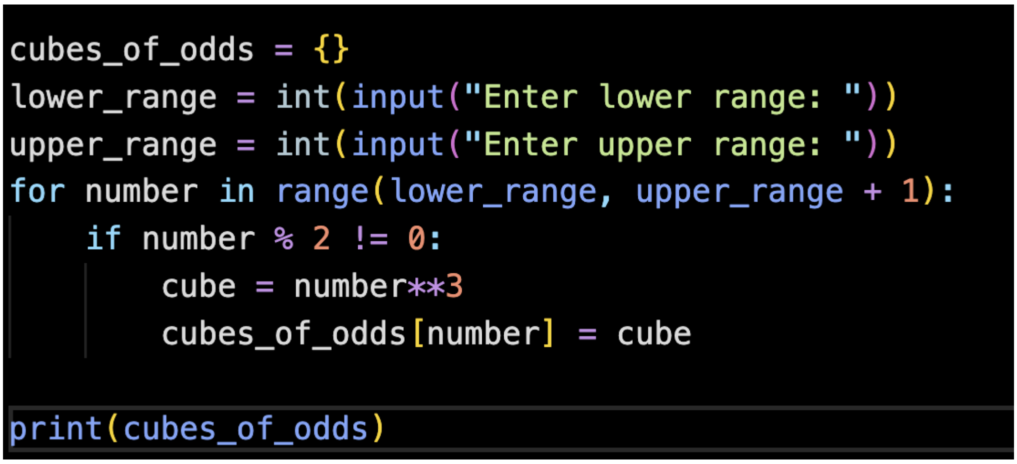
Experiment No: 2.6

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

Theory:

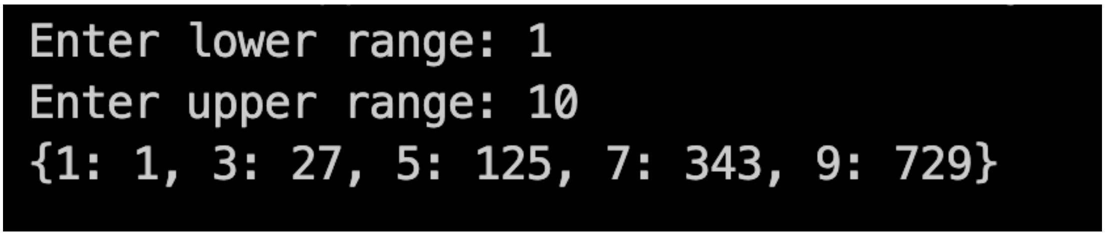
The code creates a dictionary, cubes_of_odds, storing cubes of odd numbers within a user-defined range, displaying the odd numbers and their cubes.

Code



```python
cubes_of_odds = {}
lower_range = int(input("Enter lower range: "))
upper_range = int(input("Enter upper range: "))
for number in range(lower_range, upper_range + 1):
    if number % 2 != 0:
        cube = number**3
        cubes_of_odds[number] = cube

print(cubes_of_odds)
```

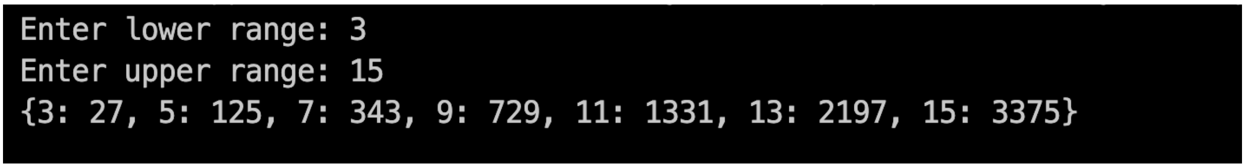
Output



```
Enter lower range: 1
Enter upper range: 10
{1: 1, 3: 27, 5: 125, 7: 343, 9: 729}
```

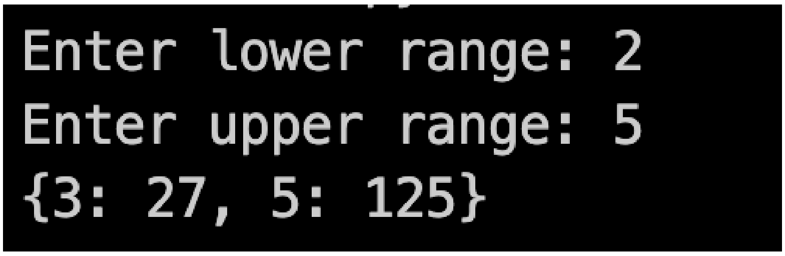
Test Case

1)



```
Enter lower range: 3
Enter upper range: 15
{3: 27, 5: 125, 7: 343, 9: 729, 11: 1331, 13: 2197, 15: 3375}
```

2)



```
Enter lower range: 2
Enter upper range: 5
{3: 27, 5: 125}
```

Conclusion: The code prompts for a range, calculates cubes for odd numbers within that range, and stores the results in a dictionary (cubes_of_odds) displaying the odd numbers with their respective cubes.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 2.7

Title: 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.

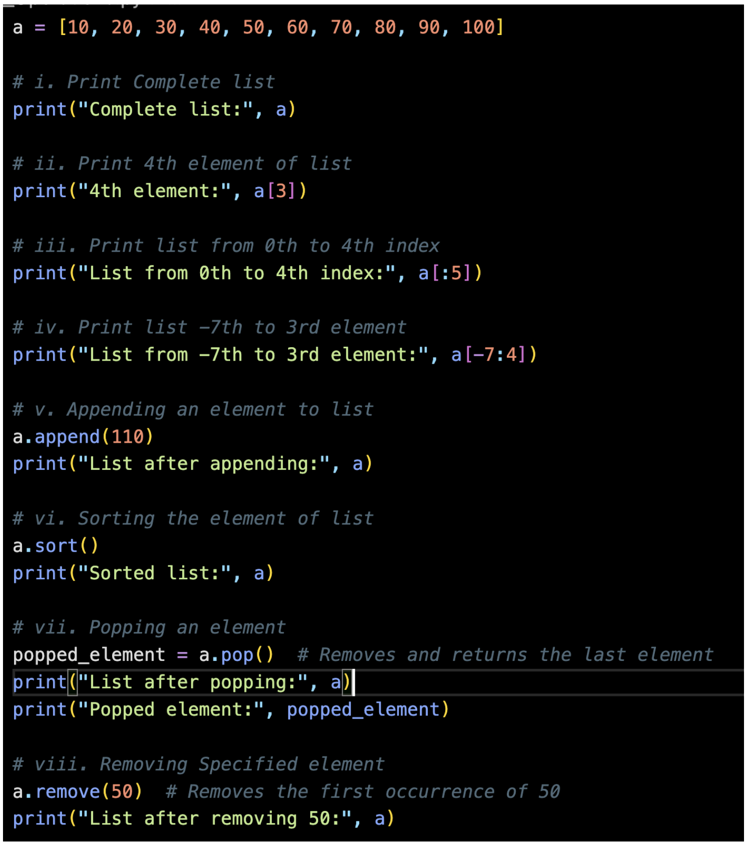
xi. Extending list.

xii. Reversing the list.

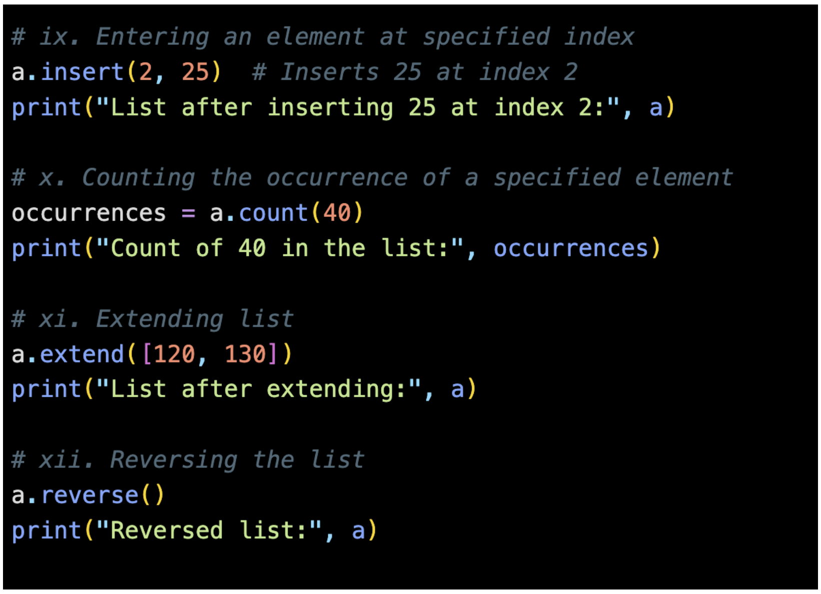
Theory:

The code demonstrates various operations on a list (a). It covers list printing, indexing, slicing, appending, sorting, popping, removing, inserting, counting, extending, and reversing elements within the list, showcasing fundamental list manipulation and retrieval methods in Python.

Code

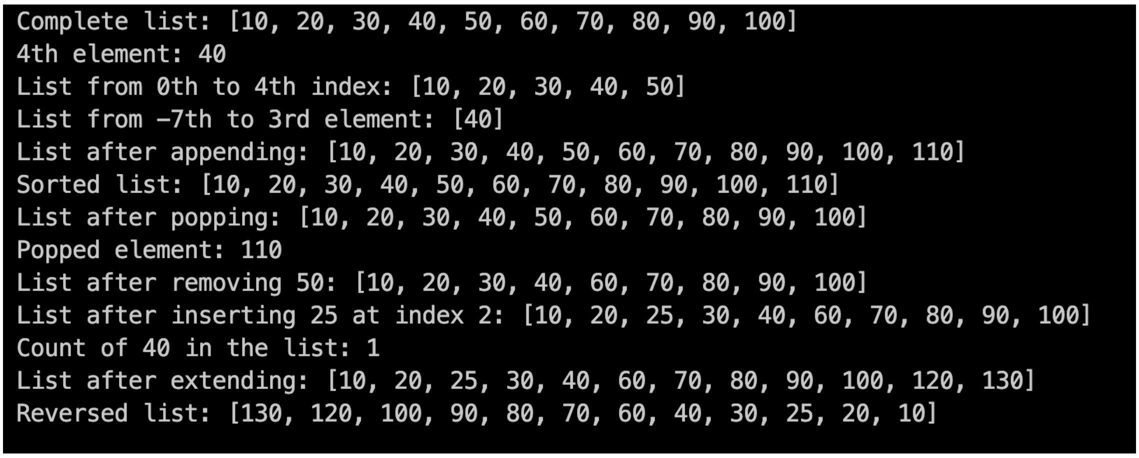


```
a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# i. Print Complete list
print("Complete list:", a)

# ii. Print 4th element of list
print("4th element:", a[3])

# iii. Print list from 0th to 4th index
print("List from 0th to 4th index:", a[:5])

# iv. Print list -7th to 3rd element
print("List from -7th to 3rd element:", a[-7:4])

# v. Appending an element to list
a.append(110)
print("List after appending:", a)

# vi. Sorting the element of list
a.sort()
print("Sorted list:", a)

# vii. Popping an element
popped_element = a.pop()  # Removes and returns the last element
print("List after popping:", a)
print("Popped element:", popped_element)

# viii. Removing Specified element
a.remove(50)  # Removes the first occurrence of 50
print("List after removing 50:", a)
```



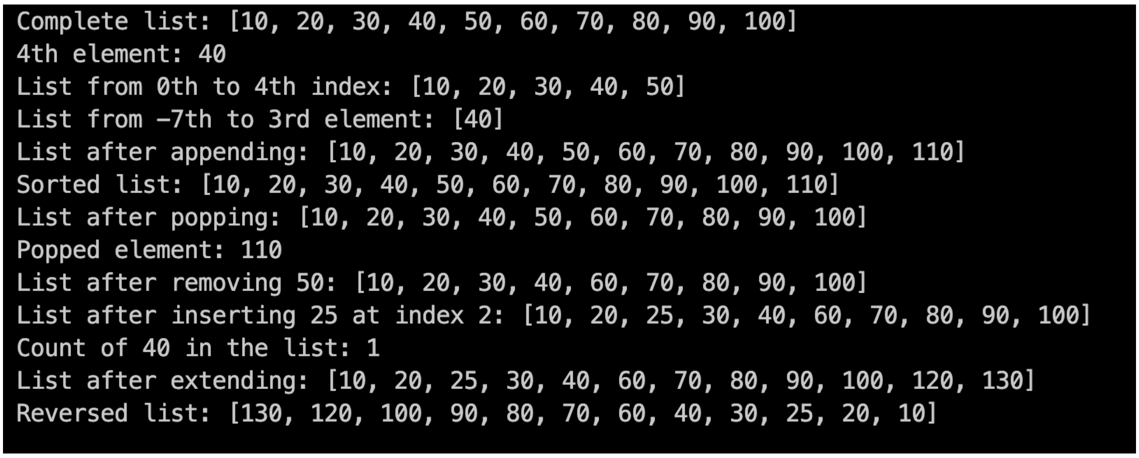
```
# ix. Entering an element at specified index
a.insert(2, 25)  # Inserts 25 at index 2
print("List after inserting 25 at index 2:", a)

# x. Counting the occurrence of a specified element
occurrences = a.count(40)
print("Count of 40 in the list:", occurrences)

# xi. Extending list
a.extend([120, 130])
print("List after extending:", a)

# xii. Reversing the list
a.reverse()
print("Reversed list:", a)
```

Output



```
Complete list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
4th element: 40
List from 0th to 4th index: [10, 20, 30, 40, 50]
List from -7th to 3rd element: [40]
List after appending: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
Sorted list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
List after popping: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Popped element: 110
List after removing 50: [10, 20, 30, 40, 60, 70, 80, 90, 100]
List after inserting 25 at index 2: [10, 20, 25, 30, 40, 60, 70, 80, 90, 100]
Count of 40 in the list: 1
List after extending: [10, 20, 25, 30, 40, 60, 70, 80, 90, 100, 120, 130]
Reversed list: [130, 120, 100, 90, 80, 70, 60, 40, 30, 25, 20, 10]
```

Test Case



```
Complete list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
4th element: 40
List from 0th to 4th index: [10, 20, 30, 40, 50]
List from -7th to 3rd element: [40]
List after appending: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
Sorted list: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
List after popping: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Popped element: 110
List after removing 50: [10, 20, 30, 40, 60, 70, 80, 90, 100]
List after inserting 25 at index 2: [10, 20, 25, 30, 40, 60, 70, 80, 90, 100]
Count of 40 in the list: 1
List after extending: [10, 20, 25, 30, 40, 60, 70, 80, 90, 100, 120, 130]
Reversed list: [130, 120, 100, 90, 80, 70, 60, 40, 30, 25, 20, 10]
```

Conclusion: The code showcases a range of list operations in Python: printing, indexing, slicing, appending, sorting, popping, removing, inserting, counting, extending, and reversing elements within the list a, demonstrating its versatile manipulation capabilities.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 3.1

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

i. **By using + operator.**

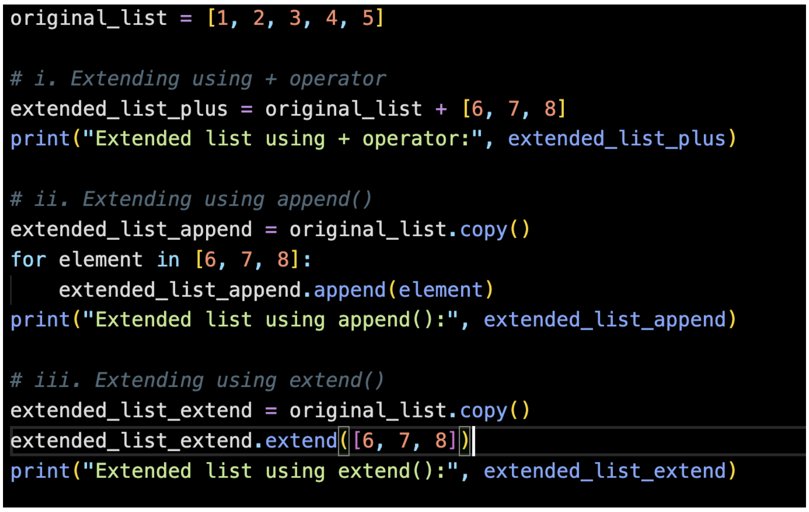
ii. **By using Append ()**

iii. **By using extend ()**

Theory:

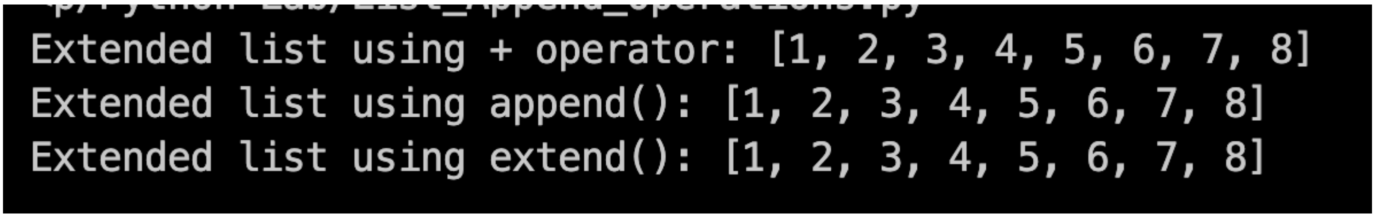
The code demonstrates different ways to extend a list in Python. It uses the + operator, append(), and extend() methods to add elements to the original list, showcasing varying approaches for list extension while preserving the original list.

Code



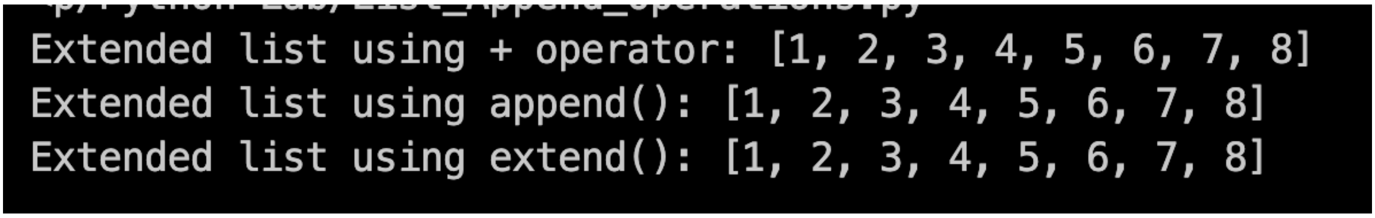
```python
original_list = [1, 2, 3, 4, 5]

# i. Extending using + operator
extended_list_plus = original_list + [6, 7, 8]
print("Extended list using + operator:", extended_list_plus)

# ii. Extending using append()
extended_list_append = original_list.copy()
for element in [6, 7, 8]:
    extended_list_append.append(element)
print("Extended list using append():", extended_list_append)

# iii. Extending using extend()
extended_list_extend = original_list.copy()
extended_list_extend.extend([6, 7, 8])
print("Extended list using extend():", extended_list_extend)
```

Output



```
Extended list using + operator: [1, 2, 3, 4, 5, 6, 7, 8]
Extended list using append(): [1, 2, 3, 4, 5, 6, 7, 8]
Extended list using extend(): [1, 2, 3, 4, 5, 6, 7, 8]
```

Test Case



```
Extended list using + operator: [1, 2, 3, 4, 5, 6, 7, 8]
Extended list using append(): [1, 2, 3, 4, 5, 6, 7, 8]
Extended list using extend(): [1, 2, 3, 4, 5, 6, 7, 8]
```

Conclusion

The code illustrates diverse techniques—using the + operator, append(), and extend() methods—to extend a list while maintaining its original content in Python.

Name of Student: Manmeet Singh

Roll Number: 16

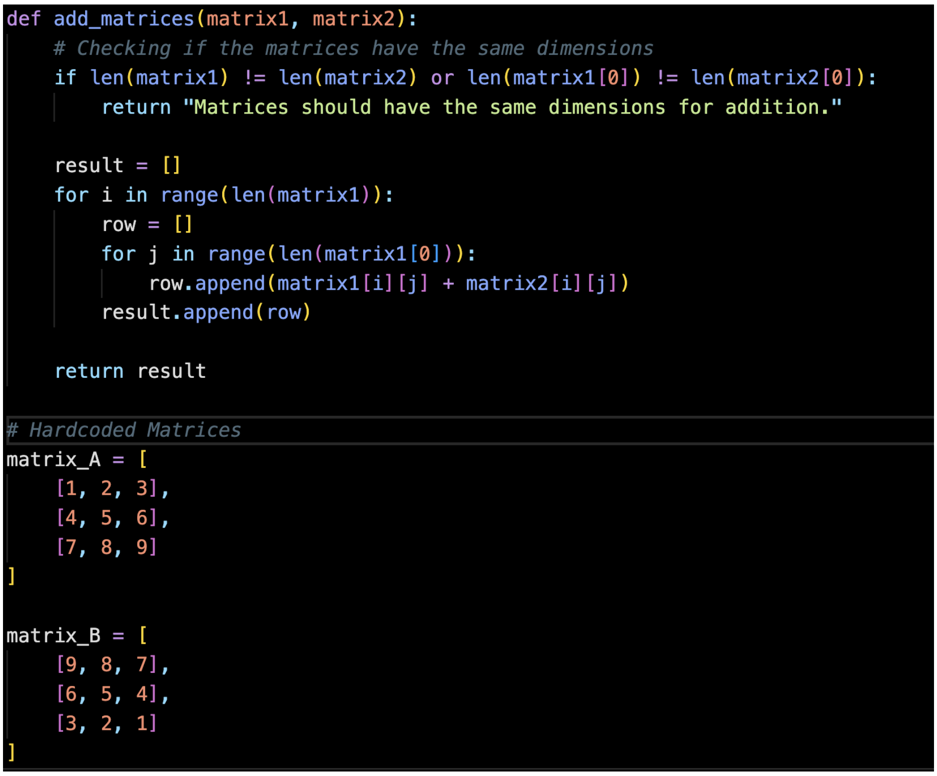
Experiment No: 3.2

Title: Write a program to add two matrices.

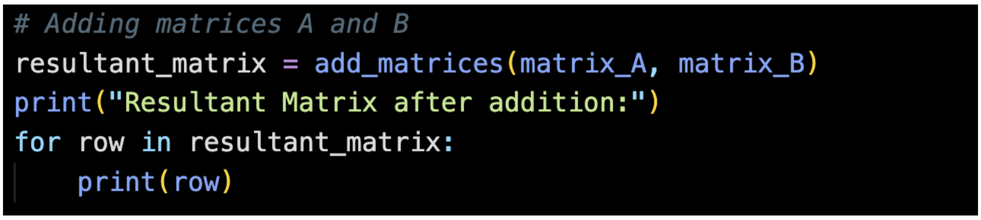
Theory:

The program defines a function add_matrices to add two matrices by iterating through their elements, checking their dimensions for compatibility, and returning the resultant matrix sum, showcasing matrix addition in Python.

Code

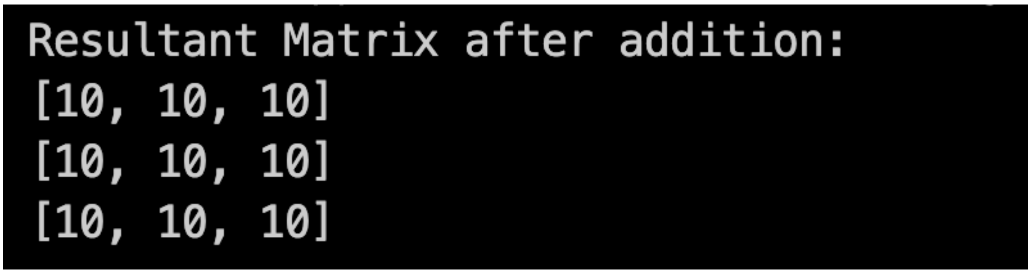


```
def add_matrices(matrix1, matrix2):
    # Checking if the matrices have the same dimensions
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        return "Matrices should have the same dimensions for addition."

    result = []
    for i in range(len(matrix1)):
        row = []
        for j in range(len(matrix1[0])):
            row.append(matrix1[i][j] + matrix2[i][j])
        result.append(row)

    return result

# Hardcoded Matrices
matrix_A = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

matrix_B = [
    [9, 8, 7],
    [6, 5, 4],
    [3, 2, 1]
]
```



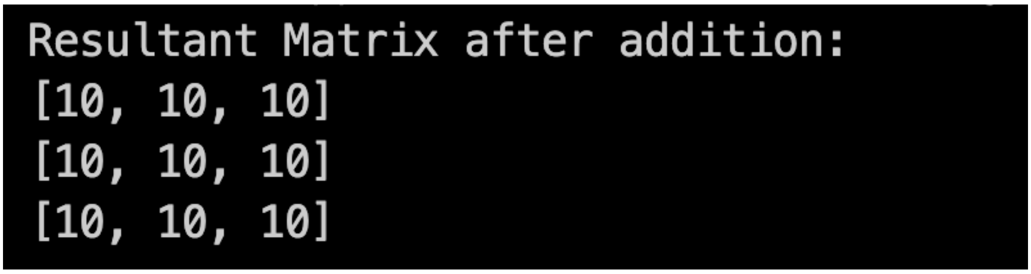
```python
# Adding matrices A and B
resultant_matrix = add_matrices(matrix_A, matrix_B)
print("Resultant Matrix after addition:")
for row in resultant_matrix:
    print(row)
```

Output



```
Resultant Matrix after addition:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

Test Case



```
Resultant Matrix after addition:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

Conclusion

The program efficiently adds two matrices, verifying their dimensions for compatibility, and displays the resultant matrix sum, demonstrating a fundamental operation in matrix mathematics using Python.

Name of Student: Manmeet Singh

Roll Number: 16

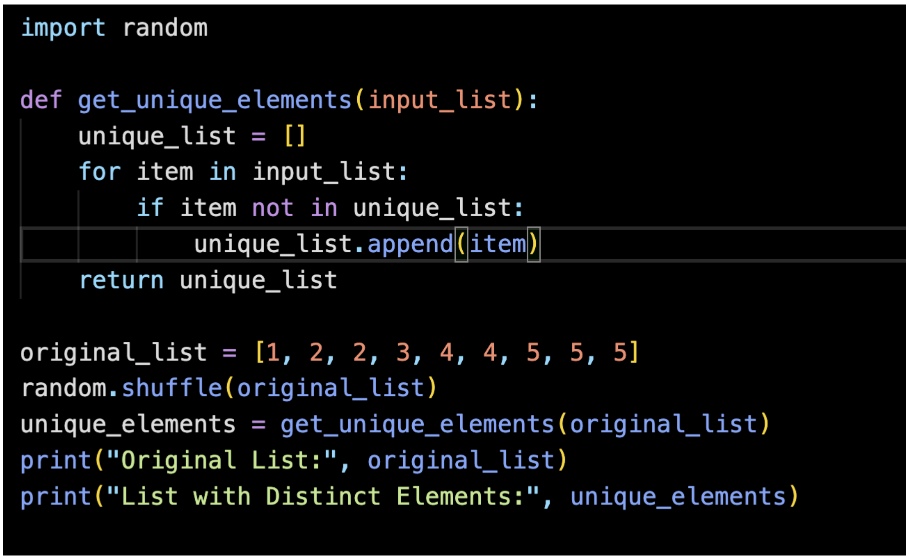
Experiment No: 3.3

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

Theory:

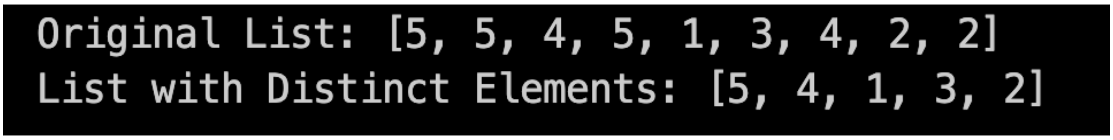
A Python function traverses a list, appends non-repeated elements to a new list, providing distinct elements without using sets or randomness.

Code



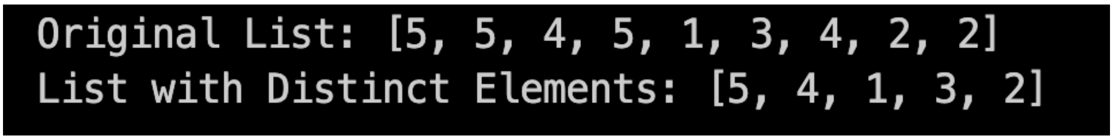
```python
import random

def get_unique_elements(input_list):
    unique_list = []
    for item in input_list:
        if item not in unique_list:
            unique_list.append(item)
    return unique_list

original_list = [1, 2, 2, 3, 4, 4, 5, 5, 5]
random.shuffle(original_list)
unique_elements = get_unique_elements(original_list)
print("Original List:", original_list)
print("List with Distinct Elements:", unique_elements)
```

Output



```
Original List: [5, 5, 4, 5, 1, 3, 4, 2, 2]
List with Distinct Elements: [5, 4, 1, 3, 2]
```

Test Case



```
Original List: [5, 5, 4, 5, 1, 3, 4, 2, 2]
List with Distinct Elements: [5, 4, 1, 3, 2]
```

Conclusion

The function efficiently extracts unique elements from a list, offering distinct values without employing sets or randomization.

Name of Student: Manmeet Singh

Roll Number: 16

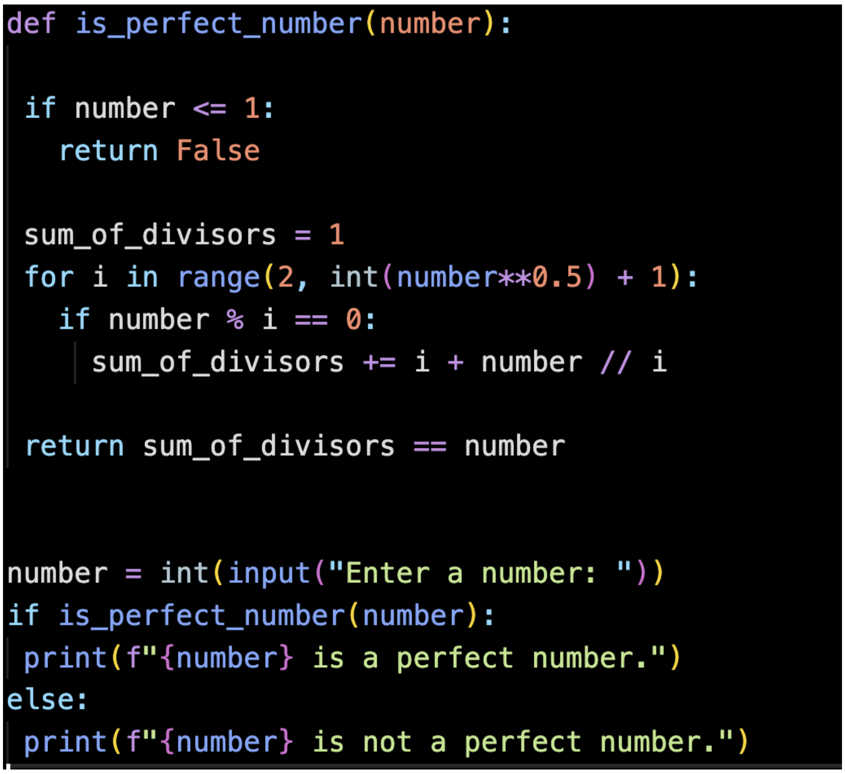
Experiment No: 3.4

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

Theory:

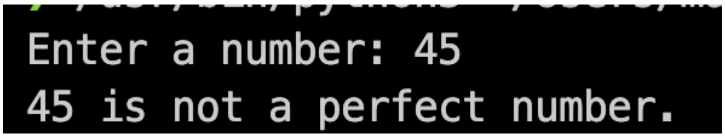
The function is_perfect_number checks if a positive integer equals the sum of its divisors (excluding itself) by iterating through divisors efficiently.

Code



```
def is_perfect_number(number):

  if number <= 1:
    return False

  sum_of_divisors = 1
  for i in range(2, int(number**0.5) + 1):
    if number % i == 0:
      sum_of_divisors += i + number // i

  return sum_of_divisors == number


number = int(input("Enter a number: "))
if is_perfect_number(number):
  print(f"{number} is a perfect number.")
else:
  print(f"{number} is not a perfect number.")
```

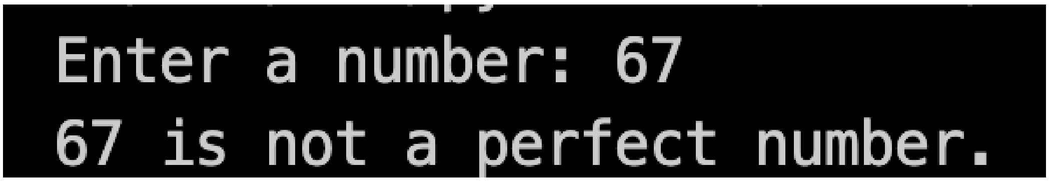
Output



```
Enter a number: 45
45 is not a perfect number.
```

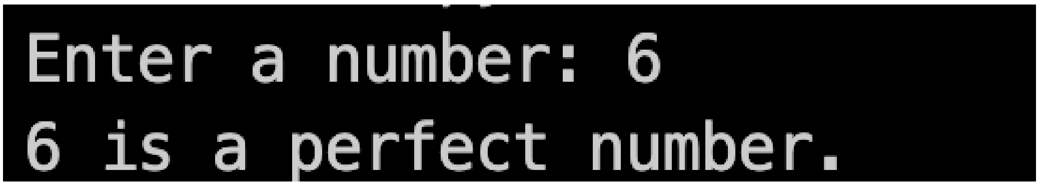
Test Case

1)



```
Enter a number: 67
67 is not a perfect number.
```

2)



```
Enter a number: 6
6 is a perfect number.
```

Conclusion

The function efficiently identifies perfect numbers by evaluating if a positive integer equals the sum of its divisors (excluding itself), providing accurate classification.

Name of Student: Manmeet Singh

Roll Number: 16

Experiment No: 3.5

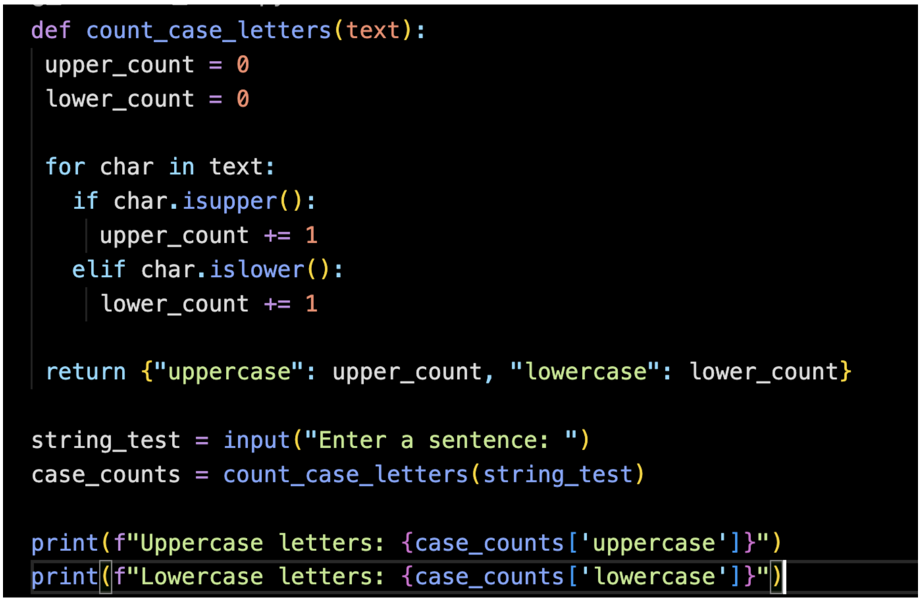
Title: Write a Python function that accepts a string and counts the number of upper and lower-case letters.

string_test= 'Today is My Best Day'

Theory:

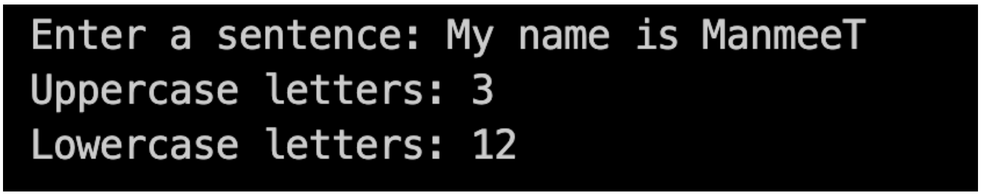
The function count_case_letters iterates through a string, counts uppercase and lowercase letters, returning the respective counts in a dictionary

Code



```python
def count_case_letters(text):
    upper_count = 0
    lower_count = 0

    for char in text:
        if char.isupper():
            upper_count += 1
        elif char.islower():
            lower_count += 1

    return {"uppercase": upper_count, "lowercase": lower_count}

string_test = input("Enter a sentence: ")
case_counts = count_case_letters(string_test)

print(f"Uppercase letters: {case_counts['uppercase']}")
print(f"Lowercase letters: {case_counts['lowercase']}")
```

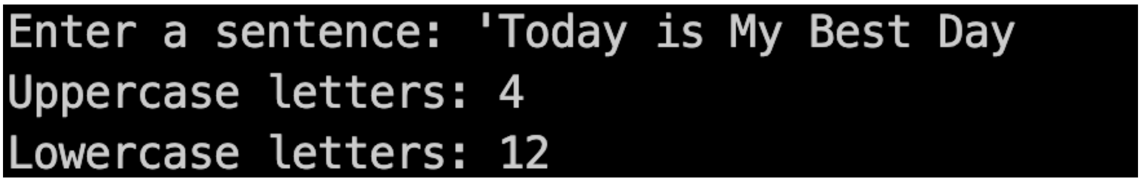
Output



```
Enter a sentence: My name is ManmeeT
Uppercase letters: 3
Lowercase letters: 12
```

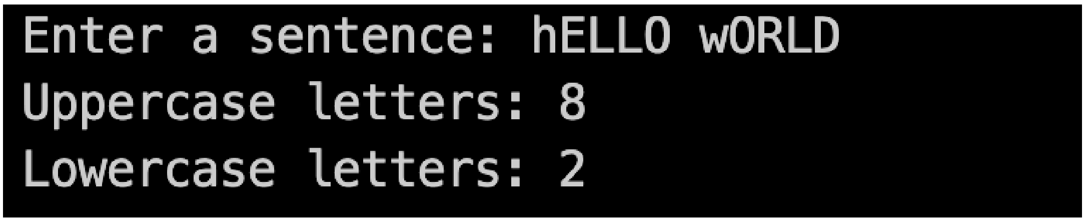
Test Cases

1)



```
Enter a sentence: 'Today is My Best Day
Uppercase letters: 4
Lowercase letters: 12
```

2)



```
Enter a sentence: hELLO wORLD
Uppercase letters: 8
Lowercase letters: 2
```

Conclusion

Efficiently counts uppercase and lowercase letters in a string, providing separate counts via a dictionary.

Name of Student: Manmeet Singh

Roll Number: 16

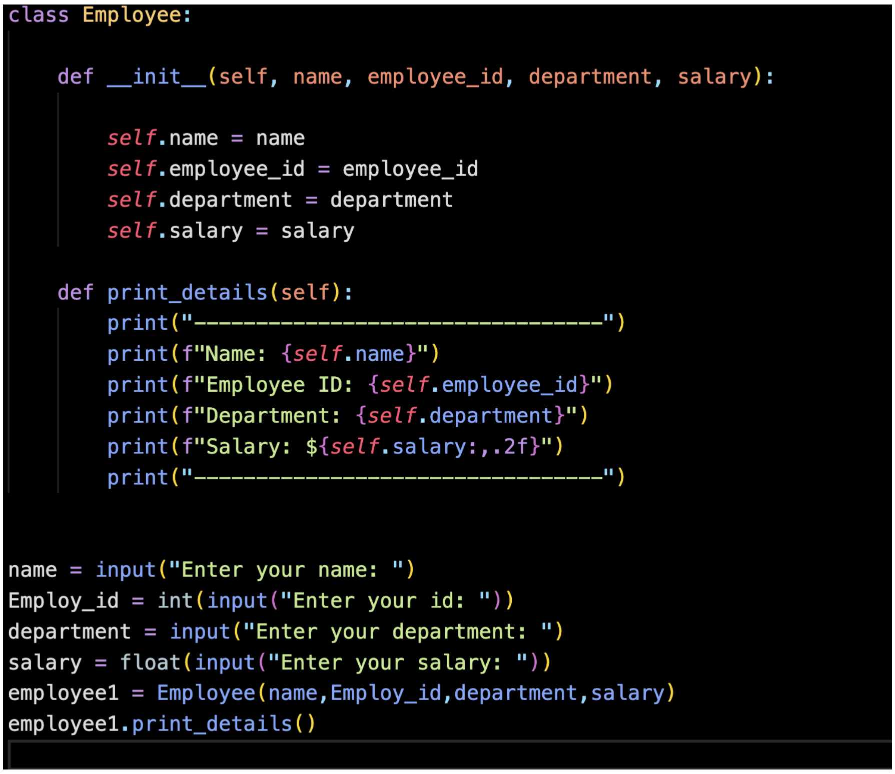
Experiment No: 4.1

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

Theory:

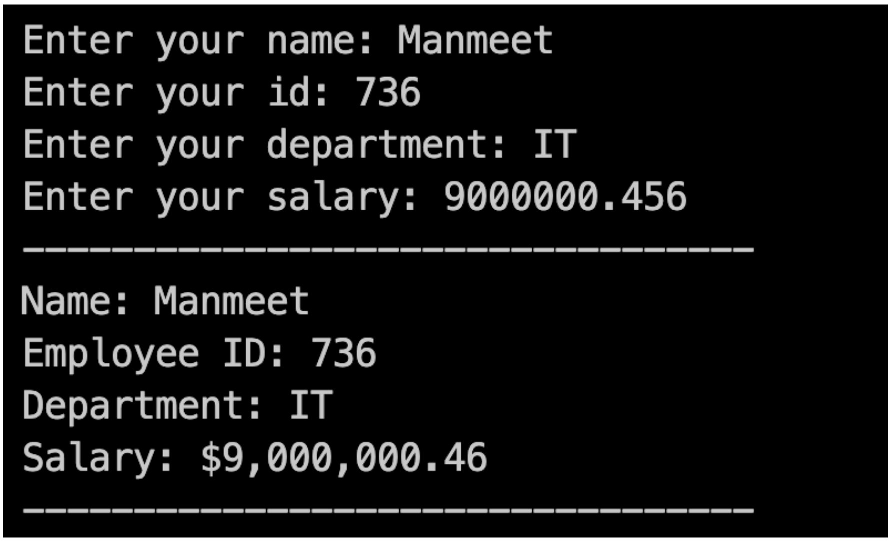
The Employee class defines attributes and a method to print employee details, instantiated to create employee instances with specified information via user input.

Code



```python
class Employee:

    def __init__(self, name, employee_id, department, salary):

        self.name = name
        self.employee_id = employee_id
        self.department = department
        self.salary = salary

    def print_details(self):
        print("---------------------------------")
        print(f"Name: {self.name}")
        print(f"Employee ID: {self.employee_id}")
        print(f"Department: {self.department}")
        print(f"Salary: ${self.salary:,.2f}")
        print("---------------------------------")


name = input("Enter your name: ")
Employ_id = int(input("Enter your id: "))
department = input("Enter your department: ")
salary = float(input("Enter your salary: "))
employee1 = Employee(name,Employ_id,department,salary)
employee1.print_details()
```

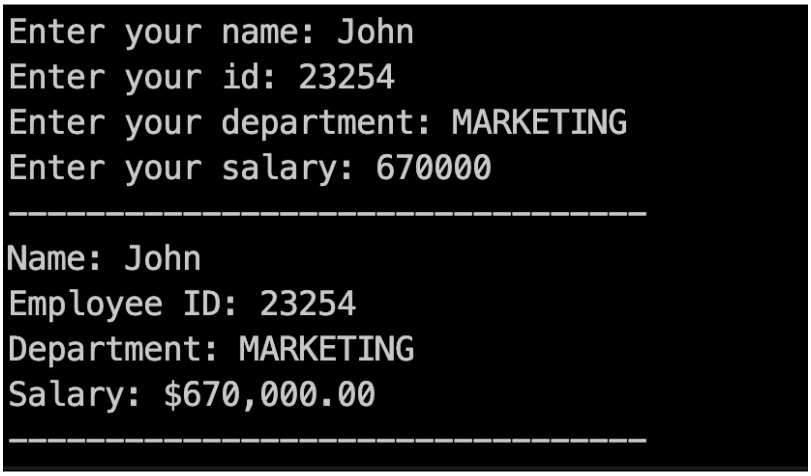
Output



```
Enter your name: Manmeet
Enter your id: 736
Enter your department: IT
Enter your salary: 9000000.456
---------------------------------
Name: Manmeet
Employee ID: 736
Department: IT
Salary: $9,000,000.46
---------------------------------
```

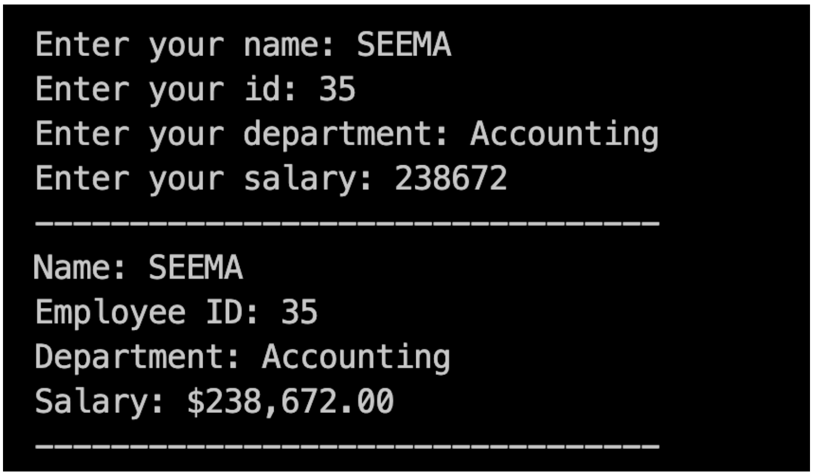
Test Cases

1)



```
Enter your name: John
Enter your id: 23254
Enter your department: MARKETING
Enter your salary: 670000
---------------------------------
Name: John
Employee ID: 23254
Department: MARKETING
Salary: $670,000.00
---------------------------------
```

2)



```
Enter your name: SEEMA
Enter your id: 35
Enter your department: Accounting
Enter your salary: 238672
---------------------------------
Name: SEEMA
Employee ID: 35
Department: Accounting
Salary: $238,672.00
---------------------------------
```

Conclusion:

It creates and displays employee details based on user input, utilizing the defined Employee class structure.

Name of Student: Manmeet Singh

Roll Number: 16

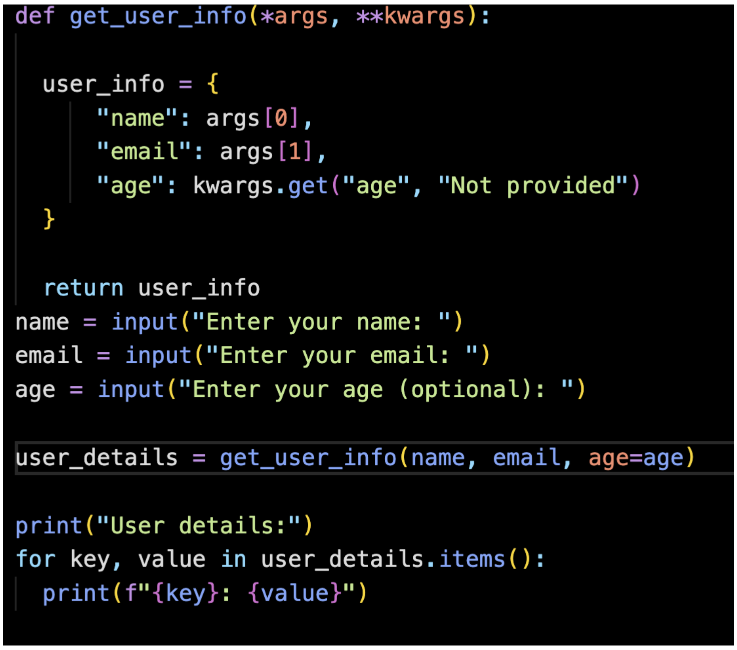
Experiment No: 4.2

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:

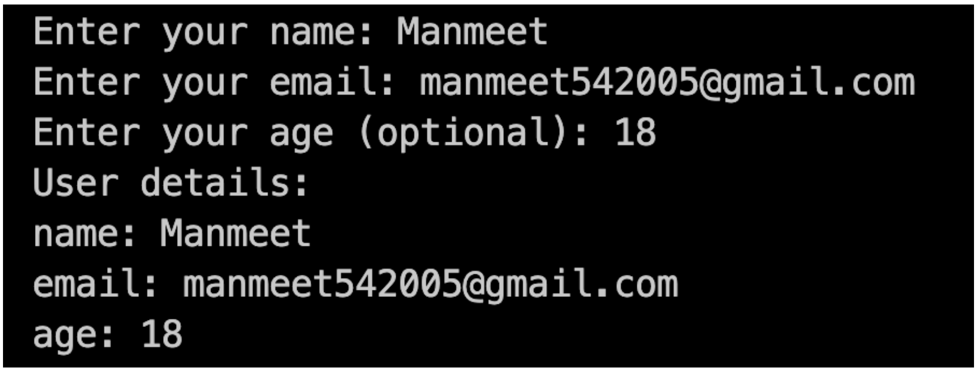
get_user_info function utilizes positional and keyword arguments to collect user details, constructing a dictionary containing name, email, and optionally age information.

Code



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

  user_info = {
      "name": args[0],
      "email": args[1],
      "age": kwargs.get("age", "Not provided")
  }

  return user_info
name = input("Enter your name: ")
email = input("Enter your email: ")
age = input("Enter your age (optional): ")

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

print("User details:")
for key, value in user_details.items():
  print(f"{key}: {value}")
```

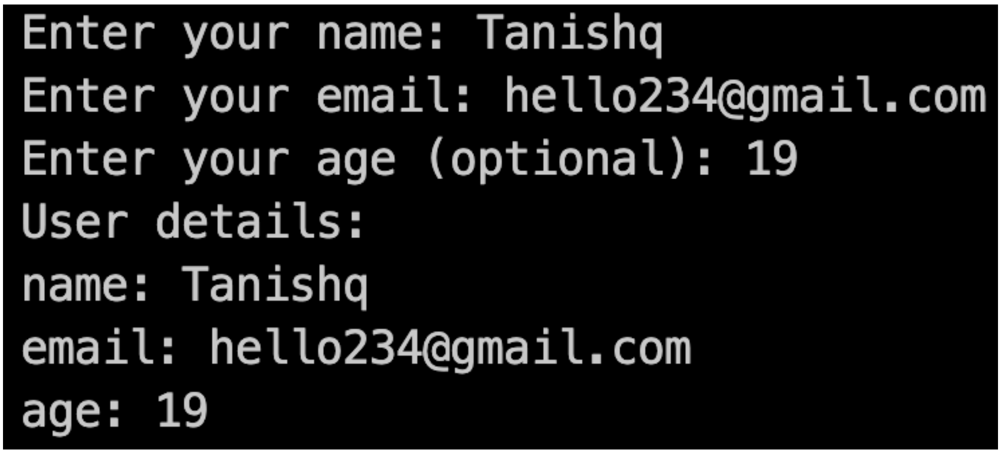
Output



```
Enter your name: Manmeet
Enter your email: manmeet542005@gmail.com
Enter your age (optional): 18
User details:
name: Manmeet
email: manmeet542005@gmail.com
age: 18
```

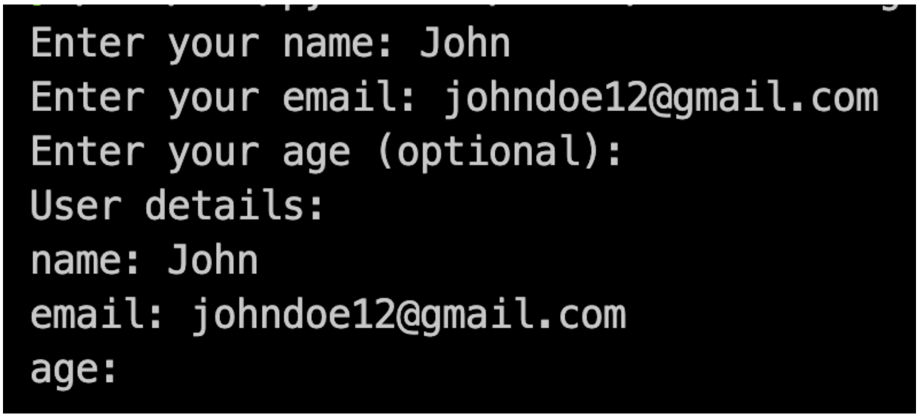
Test Cases

1)



```
Enter your name: Tanishq
Enter your email: hello234@gmail.com
Enter your age (optional): 19
User details:
name: Tanishq
email: hello234@gmail.com
age: 19
```

2)



```
Enter your name: John
Enter your email: johndoe12@gmail.com
Enter your age (optional):
User details:
name: John
email: johndoe12@gmail.com
age:
```

Conclusion

It efficiently gathers user details via a flexible argument setup, constructing a dictionary encapsulating the provided name, email, and optionally age information.

Name of Student: Manmeet Singh

Roll Number: 16

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).