Experiment 1

Program No.:1

Aim: Convert kilograms to pounds.

Software used: Collab (or) Jupiter note book.

Description:

This program converts a user-entered weight from kilograms to pounds. The program multiplies the weight in kilograms by 2.20462 (the standard conversion factor from kg to lbs). It prints the converted weight in pounds, formatted to two decimal places. If the user enters a non numeric value, a Value Error is caught, and a friendly error message is displayed.

SOURCE CODE:

```
def kg_to_pounds():
"""
Converts a weight entered in kilograms to pounds.
 """
 try:
    kilograms = float(input("Enter the weight in kilograms: "))
    pounds = kilograms * 2.20462
    print(f"The weight in pounds is: {pounds:.2f}")
 except ValueError:
    print("Invalid input. Please enter a numerical value for weight.")
# Call the function directly to run the conversion
kg_to_pounds()
```

PROCEDURE:

1. Prompt the user to enter weight in kilograms.
2. Convert the input to float and multiply by 2.20462 to get pounds.

3. Print the result rounded to two decimal places.

4. If input is invalid, display an error message.

INPUT:

Enter the weight in kilograms: 12

EXPECTED OUTPUT:

The weight in pounds is: 26.46

ACTUAL OUTPUT:

The weight in pounds is: 26.46

Program No.:1.1

Aim: Write a program that uses a for loop and while loop to print the numbers 8,11,14,…,89.

Software used: Collab (or) Jupiter note book.

Description:

This simple program uses a for loop and while loop to print a sequence of numbers starting from 8 up to (but not including) 90, incrementing by 3 each time. Range (8, 90, 3) generates numbers starting from 8 and increases by 3 each time. The loop continues until the number reaches or exceeds 90 (but does not include 90). The while loop continues as long as num_while is less than or equal to 89. Prints each number in the sequence on a new line.

SOURCE CODE:

FOR LOOP

```
print("Using a for loop:")
for num in range(8, 90, 3):
    print(num)
```

WHILE LOOP

```
print("Using a while loop:")
num_while = 8
while num_while <= 89:
print(num_while)
num_while += 3
```


PROCEDURE:

1. Print a heading: "Using a for loop or a while loop".
2. Use a for loop to iterate from 8 to 89, increasing by 3 each time.
3. Print each number in the sequence.

EXPECTED OUTPUT:

Using a while loop / For loop:

8
11
14
17
20
23
26
29
32
35
38
41
44
47
50
53
56
59
62
65
68
71
74
77
80
83
86
89

ACTUAL OUTPUT:

Using a while loop / For loop:

8

11

14

17

20

23

26

29

32

35

38

41

44

47

50

53

56

59

62

65

68

71

74

77

80

83

86

89

Program No.:1.2

Aim: Write a program to split a string into array of characters.

Software used: Collab (or) Jupyter Notebook

Description:

This simple Python program takes a string and converts it into a list of its individual characters. The string is stored in the variable my_string, and the list() function is used to split it into characters.

SOURCE CODE:

```
my_string = "venky"
 char_list = list(my_string)
 print(char_list)
```

PROCEDURE:

1. Store the string "python" in a variable.
2. Convert the string into a list of characters using list().
3. Print the resulting list.

EXPECTED OUTPUT:

['v', 'e', 'n', 'k', 'y']

ACTUAL OUTPUT:

['v', 'e', 'n', 'k', 'y']

Program 1.3:

Aim: Write a Python program to calculate the nth Fibonacci number using a function.

Software used: Collab (or) Jupyter Notebook

Description:
This program computes the nth Fibonacci number using recursion. The user enters a positive integer n, and the program returns the nth Fibonacci number. If the user enters an invalid number, an error message is displayed.

SOURCE CODE:

```
def fibonacci(n):
    if n <= 0:
        return "Input must be a positive integer"
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

n = 10
print(f"The {n}th Fibonacci number is: {fibonacci(n)}")
```


PROCEDURE:

1. Define a recursive function fibonacci(n).
2. Check if the input is valid.
3. Return base values for n = 1 or n = 2.

4. Use recursion for larger values.
5. Print the result.

INPUT:

n = 10

EXPECTED OUTPUT:

The 10th Fibonacci number is: 34

ACTUAL OUTPUT:

The 10th Fibonacci number is: 34

Program 1.4:

Aim: Write a Python program to get the largest number from a list.

Software used: Collab (or) Jupyter Notebook

Description:
This program finds the largest number in a list by comparing each element. It starts with the first element as the largest and iterates through the list to update the maximum value.

SOURCE CODE:

```
def get_largest_number(numbers):
    if not numbers:
        return None
    largest = numbers[0]
    for num in numbers[1:]:
        if num > largest:
            largest = num
    return largest

my_list = [10, 45, 2, 99, 67, 88]
largest_number = get_largest_number(my_list)
print("The largest number is:", largest_number)
```


PROCEDURE:

1. Define a function get_largest_number(numbers).
2. Initialize the first element as the largest.
3. Compare each element with the current largest.
4. Update if a larger value is found.
5. Return and print the largest number.

INPUT:

my_list = [10, 45, 2, 99, 67, 88]

EXPECTED OUTPUT:

The largest number is: 99

ACTUAL OUTPUT:

The largest number is: 99

EXPRIMENT 2

Program 2:

Aim: Write a Python program that defines a Car class with attributes like make, model, and year, and methods like start() to start the car and stop() to stop it.

Software used: Colab (or) Jupyter Notebook

Description:
This program defines a Car class with attributes such as make, model, and year. It provides methods to start and stop the car while keeping track of whether the car is currently running or not.

SOURCE CODE:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.is_started = False

    def start(self):
        if not self.is_started:
            print(f"The {self.year} {self.make} {self.model} is starting.")
            self.is_started = True
        else:
            print(f"The {self.year} {self.make} {self.model} is already running.")

    def stop(self):
        if self.is_started:
            print(f"The {self.year} {self.make} {self.model} is stopping.")
            self.is_started = False
        else:
```

```
        print(f"The {self.year} {self.make} {self.model} is already stopped.")

# Demonstrating the Car class
print("--- Car Class Demonstration ---")
my_car = Car("Toyota", "Camry", 2022)
print(f"My car: {my_car.make} {my_car.model} ({my_car.year})")

my_car.start()
my_car.start()
my_car.stop()
my_car.stop()
```

PROCEDURE:

1. Define a class Car with attributes make, model, year, and state (running or stopped).
2. Implement the method start() to start the car.
3. Implement the method stop() to stop the car.
4. Create an object of the Car class and demonstrate its methods.

INPUT:

Car("Toyota", "Camry", 2022)

EXPECTED OUTPUT:

--- Car Class Demonstration ---

My car: Toyota Camry (2022)

The 2022 Toyota Camry is starting.

The 2022 Toyota Camry is already running.

The 2022 Toyota Camry is stopping.

The 2022 Toyota Camry is already stopped.

ACTUAL OUTPUT:

--- Car Class Demonstration ---

My car: Toyota Camry (2022)

The 2022 Toyota Camry is starting.

The 2022 Toyota Camry is already running.

The 2022 Toyota Camry is stopping.

The 2022 Toyota Camry is already stopped.

Program 2.1:

Aim: Write a Python program that demonstrates inheritance by creating a base class Animal and derived classes like Dog and Cat with specific behaviors.

Software used: Colab (or) Jupyter Notebook

Description:
This program demonstrates inheritance in Python. A base class Animal is created with common behaviors (eat, sleep). Derived classes Dog and Cat extend the base class and include their own specific methods (bark, meow).

SOURCE CODE:

```
class Animal:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print(f"{self.name} is eating.")

    def sleep(self):
        print(f"{self.name} is sleeping.")

class Dog(Animal):
    def bark(self):
        print(f"{self.name} says Woof!")

class Cat(Animal):
    def meow(self):
        print(f"{self.name} says Meow!")
```

```
dog1 = Dog("Buddy")
cat1 = Cat("Whiskers")

dog1.eat()
dog1.sleep()
cat1.eat()
cat1.sleep()

dog1.bark()
cat1.meow()
```

PROCEDURE:

1. Define base class Animal with methods eat() and sleep().
2. Define derived class Dog with method bark().
3. Define derived class Cat with method meow().
4. Create objects and demonstrate inheritance.

INPUT:

Dog("Buddy"), Cat("Whiskers")

EXPECTED OUTPUT:

Buddy is eating.

Buddy is sleeping.

Whiskers is eating.

Whiskers is sleeping.

Buddy says Woof!

Whiskers says Meow!

ACTUAL OUTPUT:

Buddy is eating.

Buddy is sleeping.

Whiskers is eating.

Whiskers is sleeping.

Buddy says Woof!

Whiskers says Meow!

Program 2.2:

Aim: Write a Python program that demonstrates error handling using the try-except block to handle division by zero.

Software used: Colab (or) Jupyter Notebook

Description:
This program demonstrates exception handling in Python. It performs division safely by catching errors like division by zero and invalid data types using try-except blocks.

SOURCE CODE:

```
def safe_divide(numerator, denominator):
    try:
        result = numerator / denominator
        print(f"The result of {numerator} / {denominator} is: {result}")
    except ZeroDivisionError:
        print(f"Error: Cannot divide by zero! Attempted {numerator} / {denominator}")
    except TypeError:
        print("Error: Invalid input types. Please provide numbers.")

print("--- Error Handling Demonstration ---")
safe_divide(10, 2)
safe_divide(10, 0)
safe_divide(5, "abc")
```


PROCEDURE:

1. Define a function safe_divide() with try-except.
2. Attempt to divide two numbers.
3. Catch ZeroDivisionError if denominator is zero.
4. Catch TypeError if inputs are invalid.
5. Print results or error messages.

INPUT:

safe_divide(10, 2)

safe_divide(10, 0)

safe_divide(5, "abc")

EXPECTED OUTPUT:

The result of 10 / 2 is: 5.0

Error: Cannot divide by zero! Attempted 10 / 0

Error: Invalid input types. Please provide numbers.

ACTUAL OUTPUT:

The result of 10 / 2 is: 5.0

Error: Cannot divide by zero! Attempted 10 / 0

Error: Invalid input types. Please provide numbers.

Program 2.3:

Aim:

Write a Python program that defines a base class Animal with a method make_sound() and derived classes Dog, Cat, and Bird that override the method to produce different sounds. Demonstrate polymorphism.

Software used:

Colab (or) Jupyter Notebook

Description:
This program demonstrates polymorphism in Python. A base class Animal defines a method make_sound(). The derived classes override the method to provide their own sounds. The same method is called on different objects, showing polymorphism.

SOURCE CODE:

```
class Animal:
    def make_sound(self):
        raise NotImplementedError("Subclass must implement abstract method 'make_sound'")


class Dog(Animal):
    def make_sound(self):
        return "Woof! Woof!"


class Cat(Animal):
    def make_sound(self):
        return "Meow!"


class Bird(Animal):
    def make_sound(self):
```

```
 return "Chirp! Chirp!"


print("--- Polymorphism Demonstration ---")
animals = [Dog(), Cat(), Bird()]


for animal in animals:
    print(f"An animal makes sound: {animal.make_sound()}")
```

PROCEDURE:

1. Define base class Animal with method make_sound().
2. Define derived classes Dog, Cat, Bird that override the method.
3. Create a list of animal objects.
4. Call make_sound() for each object to demonstrate polymorphism.

INPUT:

```
animals = [Dog(), Cat(), Bird()]
```


EXPECTED OUTPUT:

```
--- Polymorphism Demonstration ---
An animal makes sound: Woof! Woof!
An animal makes sound: Meow!
An animal makes sound: Chirp! Chirp!
```

ACTUAL OUTPUT:

--- Polymorphism Demonstration ---

An animal makes sound: Woof! Woof!

An animal makes sound: Meow!

An animal makes sound: Chirp! Chirp!