Week 1

1.1 - Aim : Write a program that asks the user for a weight in kilograms and converts it to pounds.

Description : This program converts a weight given in kilograms to pounds using the conversion factor 1kg = 2.2 pounds. The program uses the standard conversion factor, multiplies the entered value by 2.2 to obtain the equivalent weight in pounds, and then displays the result.

Procedure :

1. Start the program.
2. Prompt the user to enter weight in kilograms.
3. Read and store input as float.
4. Multiply weight by 2.2.
5. Display the result.
6. End the program.

Source Code :

```
w=float(input("Enter weight in kilograms "))
pounds=2.205*w
print(f"{w} in pounds are {pounds}")
```

Input and Output:

```
============ RESTART: C:/Users/Sushma/OneDrive/Desktop/python/experments.py ============
Enter weight in kilograms: 30
30.0 kilograms is equal to 66.0 pounds.
```

Result:

The program correctly converts kilograms to pounds.

1.2 - Aim: Write a program that uses a for loop to print the numbers 8, 11, 14, 17, 20, ..., 89.

Description:

This program uses a **for loop** with the **range()** function to print numbers from 8 to 89.

It starts at 8 and increases by 3 in each iteration.

The loop continues until the value reaches 89.

Procedure:

1. Start the program.
2. Use for num in range(8,90,3) to generate numbers.
3. Print each number separated by a comma.
4. End the program.

Source Code:

```
for i in range(8,90,3):
    print(i,end= ", ")
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/experments.py ==========
8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, 53, 56, 59, 62, 65, 68, 71, 7
4, 77, 80, 83, 86, 89,
```

Result:

Numbers are correctly printed from 8 to 89 with step 3.

1.3 - Aim : Split a string into array of characters in Python.

Description:

Converts a given string into a list using the list() function.Each character of the string, including spaces, is stored as a separate element in the list. The resulting list is then printed as output.

Procedure:

1. Start the program.
2. Define the input string as s .
3. Use the list() function to convert the string s into a list of characters and store it in variable c.
4. Print the list c to display all characters of the string as individual list elements.
5. End the program

Source Code:

```
S = "Python is a programming language"
C = list(S)
print(C)
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/experments.py ==========
Array of characters: ['P', 'y', 't', 'h', 'o', 'n', ' ', 'P', 'r', 'o', 'g', 'r', 'a', '
m', 'm', 'i', 'n', 'g', ' ', 'i', 's', ' ', 'v', 'e', 'r', 'y', ' ', 'v', 'e', 'r', 'y',
' ', 'e', 'a', 's', 'y', '!']
```

Result: The string is successfully split into character.

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

Description:

This program defines a tuple containing numeric values and uses the max() function to find the largest number in the tuple. The result is then printed as the output.

Procedure:

1.Start the program.

2.Create a tuple s = (20, 7, 2, 14) containing numeric elements.

3.Use the max() function to find the largest element in the tuple. And print the max number.

4.End the program.

Source Code:

```
s=(20,7,2,14)
print("Maximum Number : ",max(s))
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/local variable.py ==========
Enter list:1 ,3 ,5,1,6,8
The largest number in the list is: 8
```

Result: The program correctly identifies the largest number.

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

Description:

This program calculates the nth Fibonacci number using a recursive function. The function fibnocci(num) calls itself to compute the sum of the two preceding numbers in the Fibonacci sequence until it reaches the base cases 0 and 1. The final result is printed as output.

Procedure :

1. Start the program.
2. Define a function fibonacci(num) to compute the Fibonacci number.
3. Check if num is 0 or 1 — return 0 or 1 respectively (base cases).
4. Otherwise, return the sum of fibonacci(num-1) and fibonacci(num-2).
5. Call the function with the desired value, e.g., fibonacci(8).
6. Print the result.
7. End the program.

Source Code :

```
def fibonacci(n):
if n <= 0:
return "Invalid input"
elif n == 1 or n == 2:
return n-1
else:
a, b = 0, 1
for _ in range(n - 2):
a, b = b, a + b
return b
n = int(input("Enter the position (n) to find nth Fibonacci number: "))
print(f"The {n}th Fibonacci number is:", fibonacci(n))
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/local variable.py ==========
Enter the position (n) to find nth Fibonacci number: 9
The 9th Fibonacci number is: 21
```

Result : The program correctly calculates the nth Fibonacci number.

WEEK 2

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

Description:

The program defines a Car class in Python with attributes make, model, and year to represent a car. It includes methods start() and stop() to simulate the car starting and stopping. Objects of the class are created to demonstrate these actions.

Procedure:

1. Define the Car class with attributes make, model, and year.
2. Create the __init__ constructor to initialize these attributes.
3. Define the start() method to display a start message.
4. Define the stop() method to display a stop message.
5. Create car objects and call their start() and stop() methods.

Source Code :

```
class Car:
def __init__(self, make, model, year):
self.make = make
self.model = model
self.year = year
def start(self):
print(f"The {self.year} {self.make} {self.model} is starting.")
def stop(self):
print(f"The {self.year} {self.make} {self.model} is stopping.")
# Example usage
car1 = Car("Toyota", "Camry", 2022)
car2=Car("Toyota", "Camry", 2032)
car1.start()
car2.stop()
```

Input and Output :

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/local variable.py ==========
The 2022 Toyota Camry is starting.
The 2032 Toyota Camry is stopping.
```

Result : Class and method execution demonstrated correctly.

2.2 - Aim: Write a Python program that demonstrates inheritance by creaϴng a base class Animal and derived classes like Dog, Cat.

Description:

The program defines a parent class Animal with a name attribute and an eat() method. The child classes Dog and Cat inherit from Animal and add their own methods (bark() and meow()). Objects of these classes demonstrate inheritance and method usage.

Procedure:

1. Define the Animal class with the __init__ constructor and eat() method.
2. Create the Dog class inheriting from Animal and add the bark() method.
3. Create the Cat class inheriting from Animal and add the meow() method.
4. Create objects of Dog and Cat with names.
5.Call the inherited eat() method and class-specific methods bark() or meow() for each object.

Source Code:

```
class Animal:
    def __init__(self, name):
        self.name = name
    def eat(self):
        print(f"{self.name} is eating.")
class Dog(Animal):
    def bark(self):
        print(f"{self.name} is barking.")
class Cat(Animal):
    def meow(self):
        print(f"{self.name} is meowing.")
d = Dog("Buddy")
c = Cat("Whiskers")
d.eat()
d.bark()
c.eat()
c.meow()
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/local variable.py ==========
Buddy is eating.
Buddy is barking.
Whiskers is eating.
Whiskers is meowing.
```

Result : Inheritance and method overriding demonstrated.

2.3 - Aim: Define a base class called Animal with a method make_sound(). Implement derived classes Dog, Cat, Bird that override make_sound() to produce different sounds.

Description: The program defines a parent class animal with a method make_sound(). The child classes dog, cat, and Bird inherit from animal and override the make_sound() method to produce their specific sounds. The program demonstrates polymorphism, where the same method name behaves differently for different objects.

Procedure:

1. Define the parent class animal with a method make_sound() for generic sounds.
2. Create child classes dog, cat, and Bird inheriting from animal.
3. Override the make_sound() method in each child class to print the respective animal sound.
4. Create a list of objects of dog, cat, and Bird.
5. Use a loop to call make_sound() on each object, demonstrating polymorphism.

Source Code:

```
class Animal:
def make_sound(self):
print("Some generic animal sound.")
class Dog(Animal):
def make_sound(self):
print("Woof!")
class Cat(Animal):
def make_sound(self):
print("Meow!")
class Bird(Animal):
def make_sound(self):
print("Tweet!")
animals = [Dog(), Cat(), Bird()]
for animal in animals:
animal.make_sound()
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/local variable.py ==========
Woof!
Meow!
Tweet!
```

Result: Polymorphism demonstrated correctly.

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

Description:

This program asks the user to input two numbers and attempts to divide the first number by the second. If the second number is zero, the program handles the error gracefully using a try-except block and displays an appropriate message instead of crashing.

Procedure:

1. Start program.
2. Prompt user for numerator and denominator.
3. Use try to perform division.
4. Use except to catch ZeroDivisionError.
5. Display result if valid.
6. End program.

Source Code:

```
        num1=int(input("Enter numerator: "))
        num2=int(input("Enter denominator: "))
        result=num1/num2
        print("Result:",result)
except ZeroDivisionError:
print("Error: Please enter valid integers.")
```

Input and Output:

```
========== RESTART: C:/Users/Sushma/OneDrive/Desktop/python/local variable.py ==========
Enter numerator: 50
Enter denominator: 2
Result: 25.0
```

Result: Program handles division by zero successfully.

WEEK 3

3.1 - Aim: Write a NumPy program using methods — info, add, array, all, greater, greater_equal, less, less_equal, equal, allclose, zeros, ones, linspace, tolist.

a. To get help on the add function

b. To test whether none of the elements of a given array is zero.

c. To create an element-wise comparison (greater, greater_equal, less and less_equal, equal, equal within a tolerance) of two given arrays.

Description:

This Python program demonstrates several basic operations and functions of the NumPy library. It includes getting help and information on functions, testing array elements, performing element-wise comparisons between arrays, and creating arrays using built-in NumPy functions like zeros(), ones(), and linspace(). It also shows how to convert a NumPy array into a Python list using tolist().

Procedure:

1. Import NumPy using import numpy as np to access its mathematical and array-handling functions.
2. Use help(np.add) and np.info(np.add) to get documentation about the add() function.
3. Create an array and use np.all() to test whether all elements are non-zero.
4. Define two arrays and perform element-wise comparisons using functions like np.greater(), np.greater_equal(), np.less(), np.less_equal(), np.equal(), and np.allclose().
5. Create arrays filled with zeros and ones using np.zeros() and np.ones().
6. Generate evenly spaced numbers using np.linspace().
7. Convert a NumPy array to a regular Python list using tolist() and display all results.

Source Code:

```
import numpy as np
a=np.array([1,2,3])
b=np.array([4,5,6])
print("Add:",np.add(a,b))
print("All non-zero:",np.all(a))
print("Greater:",np.greater(b,a))
print("Greater equal:",np.greater_equal(b,a))
print("Less:",np.less(a,b))
print("Less equal:",np.less_equal(a,b))
print("Equal:",np.equal(a,b))
```

```
print("All close:",np.allclose(a,[1,2,3]))
print("Zeros:",np.zeros(3))
print("Ones:",np.ones(3))
print("Linspace:",np.linspace(0,1,5))
print("To list:",a.tolist())
```

Input and Output:

```
Add: [5 7 9]
All non-zero: True
Greater: [ True  True  True]
Greater equal: [ True  True  True]
Less: [ True  True  True]
Less equal: [ True  True  True]
Equal: [False False False]
All close: True
Zeros: [0. 0. 0.]
Ones: [1. 1. 1.]
Linspace: [0.   0.25 0.5  0.75 1.  ]
To list: [1, 2, 3]
```

Result : Demonstrated all listed NumPy methods successfully.

3.2 - Aim : Write a NumPy program using NumPy methods — max, min, argmax, argmin, repr, count, bincount, unique.

a. To extract all numbers from a given array which are less and greater than a specified number.

b. To find the indices of the maximum and minimum numbers along the given axis of an array.

Description:

This program demonstrates the use of various NumPy methods to perform essential operations on arrays. It shows how to determine the maximum and minimum values in an array, identify their respective indices, find unique elements, and count the frequency of each element.

Procedure:

1. Start the program.
2. Import numpy as np.
3. Create a NumPy array.
4. Use np.max(), np.min() to find maximum and minimum.
5. Use np.argmax(), np.argmin() to find indices of max and min.
6. Use np.unique() to get unique elements.
7. Use np.bincount() to count occurrences of integers.
8. Display results.
9. End the program

Source Code:

```
import numpy as np
arr=np.array([1,3,2,3,4,2,1,5])
print("Max:",np.max(arr))
print("Min:",np.min(arr))
print("Argmax:",np.argmax(arr))
print("Argmin:",np.argmin(arr))
print("Unique:",np.unique(arr))
print("Bincount:",np.bincount(arr))
```

Input and Output:

```
Max: 5
Min: 1
Argmax: 7
Argmin: 0
Unique: [1 2 3 4 5]
Bincount: [0 2 2 2 1 1]
```

Result: Program demonstrates array statistics, counting, and unique elements using NumPy.

WEEK 4 - 4.1

4.1.1 - Aim : Write a Pandas program to create and display a one-dimensional array-like object containing an array of data using Pandas module.

Description:
This program demonstrates how to create a one-dimensional array-like object called a Pandas Series using the Pandas library. A Series in Pandas is similar to a one-dimensional NumPy array but can hold heterogeneous data types and comes with index labels for each element.

Procedure:

1. Start the program.
2. Import Pandas as pd.
3. Define a list of data.
4. Use pd.Series() to create a Series.
5. Print the Series.
6. End the program.

Source Code:

```
import pandas as pd
data = [18,40,17,22,19,313]
series = pd.Series(data)
print(series)
```

Input and Output :

```
0     18
1     40
2     17
3     22
4     19
5    313
dtype: int64
```

Result: A one-dimensional Pandas Series is created and displayed correctly

4.1.2 - Aim: Write a Pandas program to convert a Pandas Series to Python list and its type.

Description:

This program demonstrates how to convert a Pandas Series — a one-dimensional labeled array — into a standard Python list. The .tolist() method is used to extract the values from the Series and return them as a list. After conversion, the program also verifies the type of the resulting object using the built-in type() function.

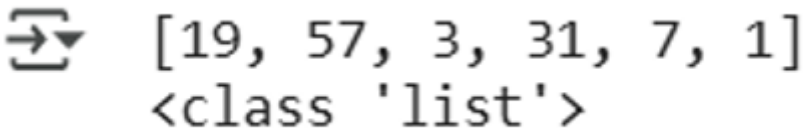
Procedure:

1. Start the program.
2. Create a Pandas Series.
3. Convert the Series to a list using .tolist().
4. Print the list and its type.
5. End the program.

Source Code:

```
import pandas as pd
data = [19,57,3,31,7,1]
series = pd.Series(data)
list_data = series.tolist()
print(list_data)
print(type(list_data))
```

Input and Output:



Result : Series is successfully converted into a Python list.

4.2

4.2.1 - Aim: Write a Pandas program to create and display a DataFrame from a specified dictionary data which has the index labels.

Description: This program demonstrates how to create a Pandas DataFrame — a two-dimensional labeled data structure — from a dictionary of data. Each key in the dictionary represents a column, and the corresponding values are the data for that column. Displaying the DataFrame allows you to view the **structured tabular data** with both column names and row indices.

Procedure:

1. Start the program.
2. Import Pandas as pd and NumPy as np.
3. Define dictionary exam_data and list labels.
4. Use pd.DataFrame(data, index=labels) to create DataFrame.
5. Print the DataFrame.
6. End the program.

Source Code:

```
import pandas as pd
import numpy as np
exam_data = {'name':
['Anastasia','Dima','Katherine','James','Emily','Michael','Matthew','Laura','Kevin','Jonas'],
'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
'attempts':[1,3,2,3,2,3,1,1,2,1],
'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels = ['a','b','c','d','e','f','g','h','i','j']
df = pd.DataFrame(exam_data,index=labels)
print(df)
```

Input and Output:

```
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

Result : DataFrame is created with specified dicϴonary data and index labels.

4.2.2

Aim : Write a Pandas program to change the name 'James' to 'Suresh' in name column

of the DataFrame.

Description:

This operation updates specific values in a pandas DataFrame column using boolean indexing. Boolean indexing allows us to select rows that meet a certain condition and modify their values.

Procedure:

1. Start program.

2. Create DataFrame as before.

3. Use df['name'].replace('James','Suresh', inplace=True) to update value.

4. Print updated DataFrame.

5. End program.

Source Code:

```
df.replace({'name': {'Dima': 'Sagar'}}, inplace=True)

df
```

Input and Output:

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Sagar	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Matthew	14.5	1	yes
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

Result : Name 'James' is successfully updated to 'Suresh'.

4.2.3 - Aim: Write a Pandas program to insert a new column in existing DataFrame.

Description:

This program demonstrates how to insert a new column into an existing pandas DataFrame. In this example, a new column named grade is added to the DataFrame, and a list of grade values is assigned to it.

Procedure:

1. Start program.
2. Create DataFrame.
3. Insert new column grade using df['grade'] = [...].
4. Print updated DataFrame.
5. End program.

Source Code:

```
df['grade'] = ['A','B','A','C','B','A','B','C','C','A']
df
```

Input and Output:

	name	score	attempts	qualify	grade
a	Anastasia	12.5	1	yes	A
b	Sagar	9.0	3	no	B
c	Katherine	16.5	2	yes	A
d	James	NaN	3	no	C
e	Emily	9.0	2	no	B
f	Michael	20.0	3	yes	A
g	Matthew	14.5	1	yes	B
h	Laura	NaN	1	no	C
i	Kevin	8.0	2	no	C
j	Jonas	19.0	1	yes	A

Result : New column is added successfully.

4.2.4 - Aim: Write a Pandas program to get list from DataFrame column headers.

Description:

This program retrieves the column headers (names) of a pandas DataFrame using the df.columns attribute. The df.columns property returns an Index object containing all column labels of the DataFrame.

Procedure:

1. Start program.
2. Create DataFrame.
3. Use list(df.columns) to get column headers.
4. Print list.
5. End program.

Source Code:

```
columns_list = list(df.columns)
print(columns_list)
```

Input and Output:

```
['name', 'score', 'attempts', 'qualify', 'grade']
```

Result : Column headers successfully converted to a list.