Appendix

LAB PROGRAMS

Contents

Program 1: Write a program to demonstrate basic data type in python

```
# Integer data type
a = 10
print("Type of a: ", type(a))
print("Value of a: ", a)

# Float data type
b = 20.5
print("Type of b: ", type(b))
print("Value of b: ", b)

# String data type
c = "Python"
print("Type of c: ", type(c))
print("Value of c: ", c)

# Boolean data type
d = True
print("Type of d: ", type(d))
print("Value of d: ", d)

# List data type
e = [1, 2, 3, 4]
print("Type of e: ", type(e))
print("Value of e: ", e)

# Tuple data type
f = (1, 2, 3, 4)
print("Type of f: ", type(f))
print("Value of f: ", f)

# Set data type
g = {1, 2, 3, 4}
print("Type of g: ", type(g))
print("Value of g: ", g)

# Dictionary data type
h = {
  "name": "Srikanth",
  "age": 30
}
print("Type of h: ", type(h))
print("Value of h: ", h)

# None data type
i = None
print("Type of i: ", type(i))
print("Value of i: ", i)
```

Output:

```
Type of a:  <class 'int'>
Value of a:  10
Type of b:  <class 'float'>
Value of b:  20.5
Type of c:  <class 'str'>
Value of c:  Python
Type of d:  <class 'bool'>
Value of d:  True
Type of e:  <class 'list'>
Value of e:  [1, 2, 3, 4]
Type of f:  <class 'tuple'>
Value of f:  (1, 2, 3, 4)
Type of g:  <class 'set'>
Value of g:  {1, 2, 3, 4}
Type of h:  <class 'dict'>
Value of h:  {'name': 'Srikanth', 'age': 30}
Type of i:  <class 'NoneType'>
Value of i:  None
```

Explanation:

In the above code, we have declared variables with different data types in Python and printed their type and value.

- 'a' is an integer type with value 10
- 'b' is a float type with value 20.5
- 'c' is a string type with value "Python"
- 'd' is a boolean type with value True
- 'e' is a list type with values [1, 2, 3, 4]
- 'f' is a tuple type with values (1, 2, 3, 4)
- 'g' is a set type with values {1, 2, 3, 4}
- 'h' is a dictionary type with values {"name": "Srikanth", "age": 30}
- 'i' is a special type None with no value.

Program 2: **Create a list and perform the following methods**

1. insert() **2. remove()** **3. append**
4. len() **5. pop()** **6. clear()**

```
# create a list
my_list = [1, 2, 3, 4, 5]
print("Original List : ", my_list)

# insert an element at a specific index
my_list.insert(2, 10)
print("List after inserting 10 at index 2 : ", my_list)

# remove an element from the list
my_list.remove(5)
print("List after removing 5 : ", my_list)

# append an element to the end of the list
my_list.append(20)
print("List after appending 20 : ", my_list)

# get the length of the list
list_length = len(my_list)
print("Length of the list : ", list_length)

# remove the last element from the list
last_element = my_list.pop()
print("Last element removed from the list : ", last_element)
print("List after removing the last element : ", my_list)

# clear all elements from the list
my_list.clear()
print("List after clearing all elements : ", my_list)
```

Output:

```
Original List :  [1, 2, 3, 4, 5]
List after inserting 10 at index 2 :  [1, 2, 10, 3, 4, 5]
List after removing 5 :  [1, 2, 10, 3, 4]
List after appending 20 :  [1, 2, 10, 3, 4, 20]
Length of the list :  6
Last element removed from the list :  20
List after removing the last element :  [1, 2, 10, 3, 4]
List after clearing all elements :  []
```

Explanation:

In the above code, we have created a list my_list with elements [1, 2, 3, 4, 5]. We perform the following operations on the list:

- The insert() method is used to insert an element at a specific index. In this case, we have inserted 10 at index 2.
- The remove() method is used to remove an element from the list. In this case, we have removed 5 from the list.
- The append() method is used to append an element to the end of the list. In this case, we have appended 20 to the end of the list.
- The len() function is used to get the length of the list.
- The pop() method is used to remove the last element from the list.
- The clear() method is used to remove all elements from the list.

Program 3: **Create a tuple and perform the following methods**

1. Add items 2. len() 3. check for item in tuple 4. Access items

```
# Create a tuple
my_tuple = (1, 2, 3, 4, 5)
print("Original Tuple: ", my_tuple)

# Get the length of the tuple
tuple_length = len(my_tuple)
print("Length of the Tuple: ", tuple_length)

# Check if an item exists in the tuple
check = 2 in my_tuple
print("Is 2 in the Tuple?: ", check)

# Access items in the tuple
print("Item at index 2: ", my_tuple[2])

# Convert the tuple to a list as tuples are immutable
my_list = list(my_tuple)
```

```
# Insert an element at a specific index in the list
my_list.insert(2, 20)

# Append an element to the end of the list
my_list.append(6)

# Convert the list back to a tuple
my_tuple = tuple(my_list)

# Print the updated tuple
print("Elements in Tuple: ", my_tuple)
```

Output:

```
Original Tuple:  (1, 2, 3, 4, 5)
Length of the Tuple:  5
Is 2 in the Tuple?:  True
Item at index 2:  3
Elements in Tuple:  (1, 2, 20, 3, 4, 5, 6)
```

Explanation:

In the above code, we have created a tuple my_tuple with elements (1, 2, 3, 4, 5). We perform the following operations on the tuple:

- The len() function is used to get the length of the tuple.
- The in operator is used to check if an item exists in the tuple. In this case, we check if 2 exists in the tuple.
- Tuples are ordered, and we can access elements in a tuple using indices. In this case, we access the element at index 2 in the tuple.

Since tuples are immutable, we can't add elements to them directly. To add elements to a tuple, we first convert the tuple to a list using the list() function. Then, we can use the insert() method to insert an element at a specific index and the append() method to add an element to the end of the list. Finally, we convert the list back to a tuple using the tuple() function. The updated tuple is printed with the print() statement.

Program 4: **Create a dictionary and apply the following methods**

1. Print the dictionary items **2. access items** **3. use get()**

4. change values **5. use len()**

```
# Create a dictionary
student = {"name": "Mary", "age": 20, "grade": "A"}

# Print the dictionary items directly
print("Dictionary items using items() : \n", student.items())

# Print the dictionary items by traversing it
print("Dictionary items by traversing :")
for key, value in student.items():
    print(key, ":", value)

# Access items in the dictionary using keys
print("Age of student : ", student["age"])
```

```
# Use the get() method to access items
print("Grade of student : ", student.get("grade"))

# Change values in the dictionary
student["grade"] = "B"
print("Updated dictionary : ", student)

# Get the length of the dictionary
dict_length = len(student)
print("Length of the dictionary : ", dict_length)
```

Output:

```
Dictionary items using items() :
dict_items([('name', 'Mary'), ('age', 20), ('grade', 'A')])
Dictionary items by traversing :
name : Mary
age : 20
grade : A
Age of student :  20
Grade of student :  A
Updated dictionary :  {'name': 'Mary', 'age': 20, 'grade': 'B'}
```

Explanation:

- In the above code, we create a dictionary student with keys name, age, and grade and corresponding values.
- We print the items in the dictionary directly using the .items() method.
- We use a for loop to traverse the dictionary and print each item in the dictionary.
- We access the values in the dictionary using both the square bracket notation and the .get() method.
- We change the value of grade key in the dictionary.
- Finally, we find the length of the dictionary using the len() function.

Program 5: **Write a program to create a menu with the following options**
1. TO PERFORM ADDITION **2. TO PERFORM SUBTRACTION**
3. TO PERFORM MULTIPLICATION **4. TO PERFORM DIVISION**
Accepts users input and perform the operation accordingly. Use functions with arguments.

```
# Define functions for operations
def add(a, b):
    return a + b
```

```
def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    return a / b

# Create menu
while True:
    print("ARITHMETIC OPERATIONS")
    print("*****************************")
    print("1. TO PERFORM ADDITION")
    print("2. TO PERFORM SUBTRACTION")
    print("3. TO PERFORM MULTIPLICATION")
    print("4. TO PERFORM DIVISION")
    print("5. EXIT")
    choice = int(input("Enter your choice: "))
    if choice == 5:
        print("Exiting the program")
        break

    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))

    if choice == 1:
        result = add(num1, num2)
        print("Result: ", result)

    elif choice == 2:
        result = subtract(num1, num2)
        print("Result: ", result)
    elif choice == 3:
        result = multiply(num1, num2)
        print("Result: ", result)
    elif choice == 4:
        result = divide(num1, num2)
        print("Result: ", result)
    else:
        print("Invalid option, please try again")
```

Output:

```
ARITHMETIC OPERATIONS
*****************************
1. TO PERFORM ADDITION
2. TO PERFORM SUBTRACTION
3. TO PERFORM MULTIPLICATION
4. TO PERFORM DIVISION
5. EXIT
Enter your choice: 1
Enter first number: 10
Enter second number: 20
Result:  30
ARITHMETIC OPERATIONS
*****************************
1. TO PERFORM ADDITION
2. TO PERFORM SUBTRACTION
3. TO PERFORM MULTIPLICATION
4. TO PERFORM DIVISION
5. EXIT
Enter your choice: 2
Enter first number: 10
Enter second number: 20
Result:  -10
ARITHMETIC OPERATIONS
*****************************
1. TO PERFORM ADDITION
2. TO PERFORM SUBTRACTION
3. TO PERFORM MULTIPLICATION
4. TO PERFORM DIVISION
5. EXIT
Enter your choice: 3
Enter first number: 10
Enter second number: 20
Result:  200

ARITHMETIC OPERATIONS
*****************************
1. TO PERFORM ADDITION
2. TO PERFORM SUBTRACTION
3. TO PERFORM MULTIPLICATION
4. TO PERFORM DIVISION
5. EXIT
Enter your choice: 4
Enter first number: 10
Enter second number: 20
Result:  0.5
ARITHMETIC OPERATIONS
*****************************
1. TO PERFORM ADDITION
2. TO PERFORM SUBTRACTION
3. TO PERFORM MULTIPLICATION
4. TO PERFORM DIVISION
5. EXIT
Enter your choice: 5
Exiting the program
```

Explanation:

The program is a simple calculator which performs basic arithmetic operations (addition, subtraction, multiplication, division). It offers the user a menu of options to choose from and then accepts the inputs. The program uses functions for each operation with arguments and calls the respective function based on the user's choice. It continues until the user presses option 5 to exit. If the user enters an invalid option, it displays an error message.

Program 6: **Write a python program to print a number is positive/negative using if-else.**

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

# Check if the number is positive or negative
if num >= 0:
    print(num, "is a positive number")
else:
    print(num, "is a negative number")
```

Output:

```
Enter a number: 6
6 is a positive number

Enter a number: -25
-25 is a negative number
```

Explanation:

- The input() method is used to accept input from the user in the form of a string.
- The int() is used to convert the string to an integer.
- if checks if the number is greater than or equal to zero, and if so, it prints "is a positive number". Else executes if the number is less than zero and prints "is a negative number".

Program 7: **Write a program for filter() to filter only even numbers from a given list.**

```
# Input list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Define a function to check for even numbers
def is_even(num):
    return num % 2 == 0

# Use filter() to filter only even numbers
even_numbers = list(filter(is_even, numbers))

# Print the filtered list of even numbers
print("The even numbers are: ", even_numbers)
```

Output:

```
The even numbers are:  [2, 4, 6, 8, 10]
```

Explanation:

- The variable 'numbers' is a list of integers.
- The is_even() function takes an integer as an argument and returns True if it is even, and False if it is odd.
- The filter() method takes two arguments, the first is a function and the second is an iterable (in this case, the list numbers). It filters out elements from the list that return False when passed to the function.

- The list() function is used to convert the filtered list of even numbers from a filter object to a list.
- The filtered list of even numbers is then printed using print().

Program 8: **Write a program to print date, time for today and now**

```
import datetime

# Get the current date and time
now = datetime.datetime.now()

# Print the date and time
print("Today's date and time:", now)
```

Output:

```
Today's date and time: 2023-02-01 23:33:35.938836
```

Explanation:

- The import datetime imports the datetime module, which provides classes to work with dates and times.
- The statement datetime.datetime.now() returns the current date and time as a datetime object.
- The print() function is used to print the current date and time, which is stored in the variable now.

Program 9: **Write a python program to add some days to your present date and print the date added.**

```
import datetime

# Get the current date
today = datetime.datetime.now().date()

# Print the current date
print("Current Date is :", today)

# Input number of days to add
days_to_add = int(input("Enter the number of days to add : "))

# Add the days to the current date
new_date = today + datetime.timedelta(days=days_to_add)

# Print the new date
print("Date after adding", days_to_add, "days is :", new_date)
```

Output:

```
Current Date is : 2023-02-06
Enter the number of days to add : 5
Date after adding 5 days is : 2023-02-11
```

Explanation:

This code calculates and prints the date after adding a specified number of days to the current date. Here's a more detailed explanation of each step:

- **Importing the datetime module:** The first line of the code imports the datetime module from the Python Standard Library. This module provides classes for working with dates and times.
- **Getting the current date:** The line today = datetime.datetime.now().date() gets the current date as a date object using the now() method from the datetime class. The date() method is then called on the datetime object to extract only the date component. The result is stored in the today variable.
- **Printing the current date:** The line print("Current Date is :", today) prints the current date stored in the today variable.
- **Entering the number of days to add:** The line days_to_add = int(input("Enter the number of days to add : ")) prompts the user to enter the number of days to add. The input() function returns the entered value as a string, which is then converted to an integer using the int() function and stored in the days_to_add variable.
- **Calculating the new date:** The line new_date = today + datetime.timedelta(days=days_to_add) calculates the new date by creating a timedelta object that represents days_to_add days using the timedelta class from the datetime module. The timedelta object is then added to the today variable to get the new date, which is stored in the new_date variable.
- **Printing the new date:** The line print("Date after adding ", days_to_add, "days is :", new_date) prints the new date stored in the new_date variable.

Program 10: **Write a program to count the number of occurrences of each character in a string and store the result in a dictionary data structure:**

```
def count_characters(string):
    # Create an empty dictionary
    char_count = {}

    # Loop through each character in the string
    for char in string:
        # If the character is already in the dictionary, increment its count by 1
        if char in char_count:
            char_count[char] += 1
        # If the character is not in the dictionary, add it and set its count to 1
        else:
            char_count[char] = 1

    # Return the dictionary
    return char_count

# Input the string
string = input("Enter a string: ")
```

Output:

```
Enter a string: MalayalaM
Character count in 'MalayalaM' is :
{'M': 2, 'a': 4, 'l': 2, 'y': 1}
```

```
# Call the function and store the result in a variable
result = count_characters(string)

# Print the result
print("Character count in '" + string + "' is :\n" + str(result))
```

Explanation:

The count_characters function takes a string as input and returns a dictionary that maps each character in the string to its count. The function creates an empty dictionary char_count, loops through each character in the string, and either increments the count of the character if it's already in the dictionary, or adds the character to the dictionary and sets its count to 1 if it's not. Finally, the function returns the char_count dictionary.

Program 11: **Write a program to count frequency of characters in a given file.**

```
def count_characters(filename):
    # Open the file in read mode
    with open(filename, "r") as file:
        # Read the contents of the file into a string
        contents = file.read()

        # Create an empty dictionary
        char_count = {}

        # Loop through each character in the string
        for char in contents:
            # If the character is already in the dictionary, increment its count by 1
            if char in char_count:
                char_count[char] += 1
            # If the character is not in the dictionary, add it and set its count to 1
            else:
                char_count[char] = 1

        # Return the dictionary
        return char_count

# Input the data
data = input("Enter the data: ")

# Input the filename to be saved
filename = input("Enter the filename to save the data: ")

# Write the data to the file
with open(filename, "w") as file:
    file.write(data)
```

Output:

```
Enter the data: Mississippi
Enter the filename to save the data: myData.txt
Opening the file 'myData.txt' for reading...
Character frequency in 'myData.txt' is :
{'M': 1, 'i': 4, 's': 4, 'p': 2}
```

```
# Inform that the file is being opened for reading
print("Opening the file '" + filename + "' for reading...")

# Call the function and store the result in a variable
result = count_characters(filename)

# Print the result
print("Character frequency in '" + filename + "' is :\n" + str(result))
```

Explanation:

- The given program counts the frequency of characters in a file. It starts by asking the user to enter the data that needs to be stored in the file. The user is then prompted to enter the name of the file where the data needs to be saved. The data is then stored in the file using the write method of the file object.
- Once the data is stored in the file, the program informs the user that it is opening the file for reading and calls the count_characters function. This function takes the filename as an argument and opens the file in read mode using the open method. The contents of the file are then read into a string using the read method. The function then creates an empty dictionary and loops through each character in the string. If a character is already in the dictionary, its count is incremented by 1. If the character is not in the dictionary, it is added and its count is set to 1. Finally, the function returns the dictionary which stores the frequency of each character in the file. The result is then printed to the console.

Program 12: **Using a numpy module create an array and check the following**
1. Type of array **2. Axes of array**
3. Shape of array **4. Type of elements in array.**

```
import numpy as np

# Create a one-dimensional array
arr = np.array([1, 2, 3, 4, 5])

# Print the type of the array
print("Type of array:", type(arr))

# Print the number of axes of the array
print("Axes of array:", np.ndim(arr))

# Print the shape of the array
print("Shape of array:", arr.shape)

# Print the type of elements in the array
print("Type of elements in array:", arr.dtype)
```

Output:

```
Type of array: <class 'numpy.ndarray'>
Axes of array: 1
Shape of array: (5,)
Type of elements in array: int32
```

Explanation:

- The given program is used to create an array using the NumPy module in Python and check various attributes of the array. NumPy is a library for the Python programming language that is used for scientific computing and data analysis. It provides functions for creating and manipulating arrays, which are multi-dimensional collections of numerical data.
- In the program, the first attribute that is checked is the "type of array." This refers to the class of the object created by the NumPy np.array method. In this case, the type of the array will be numpy.ndarray, which is the class that represents arrays in NumPy.
- The next attribute checked is the "axes of array." The number of axes of an array refers to the number of dimensions in the array. In NumPy, an array can have one or more axes, where each axis represents a separate dimension. In this program, the number of axes is found using the np.ndim function, which returns the number of dimensions of the input array. In general, when the array has more than one dimension, it is called a multi-dimensional array, and the number of dimensions is equal to the number of axes.
- The third attribute checked is the "shape of array." The shape of an array refers to the size of the array along each of its axes. In this program, the shape of the array is found using the shape attribute of the arr object. This returns a tuple that represents the size of the array along each of its axes.
- Finally, the "type of elements in array" is checked. This refers to the data type of the elements stored in the array. In NumPy, arrays can contain elements of different data types, such as integers, floats, or strings. In this program, the type of elements in the array is found using the dtype attribute of the arr object. This returns the data type of the elements in the array, such as int32 or float64.

Program 13: Write a python program to concatenate the dataframes with two different objects.

```
import pandas as pd

# Create a first DataFrame df1
df1 = pd.DataFrame({'Name':['Snigdha','Smayan','Satvik'],
                    'age':[20,19,18],
                    'grade':['A','C','A']},
                   index=[1,2,3])

# Create a second DataFrame df2
df2 = pd.DataFrame({'Name':['Mary','Nirmala','Shantala'],
                    'age':[23,30,28],
                    'grade':['B','C','A']},
                   index=[4,5,6])

# Concatenate the two DataFrames into a single DataFrame
result = pd.concat([df1, df2])

# Display the concatenated DataFrame
print(result)
```

Output:

```
      Name  age grade
1   Snigdha   20     A
2    Smayan   19     C
3    Satvik   18     A
4      Mary   23     B
5   Nirmala   30     C
6  Shantala   28     A
```

Explanation:

- **Meaning and Purpose of DataFrame in Pandas:** A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. It is a primary data structure in the Pandas library, used for data manipulation and analysis. The purpose of a DataFrame is to store and manipulate tabular data, such as rows and columns,
- The above program utilizes the Pandas library to demonstrate the creation and manipulation of two DataFrames, df1 and df2. The pd.DataFrame() function is used to create each DataFrame, with a dictionary of data passed as an argument.
- Each DataFrame contains columns for names, ages, and grades, with the index specified for each row. The two DataFrames are then concatenated using the pd.concat() function, which merges the two DataFrames into a single DataFrame, result.
- The print() function is used to display the resulting concatenated DataFrame.
- In general, DataFrames are a crucial data structure in Pandas, used for storing and analyzing tabular data with labeled rows and columns.

Program 14: **Write a python code to read a csv file using pandas module and print the first and last five lines of a file.**

```
import pandas as pd

# Read the csv file into a DataFrame using pd.read_csv()
df = pd.read_csv("https://people.sc.fsu.edu/~jburkardt/data/csv/hw_200.csv")

# Display the first 5 rows of the DataFrame using head()
print("First 5 Rows:")
print(df.head(5))

# Display the last 5 rows of the DataFrame using tail()
print("\nLast 5 Rows:")
print(df.tail(5))
```

Output:

```
First 5 Rows:
   Index   Height(Inches)"   "Weight(Pounds)"
0      1             65.78            112.99
1      2             71.52            136.49
2      3             69.40            153.03
3      4             68.22            142.34
4      5             67.79            144.30

Last 5 Rows:
     Index   Height(Inches)"   "Weight(Pounds)"
195    196             65.80            120.84
196    197             66.11            115.78
197    198             68.24            128.30
198    199             68.02            127.47
199    200             71.39            127.88
```

Explanation:

- The above program uses the Pandas library to read a csv file located at https://people.sc.fsu.edu/~jburkardt/data/csv/hw_200.csv. The pd.read_csv() function is used to read the csv file into a Pandas DataFrame, which is a 2-dimensional labeled data structure that is used to store and manipulate data in a tabular form.
- This URL contains a csv file with data about human body weight, including the person's index number, their weight in pounds, and their height in inches.
- Once the csv file has been read into the DataFrame, the first 5 rows of the DataFrame are displayed using the head() function and the last 5 rows of the DataFrame are displayed using the tail() function. These functions return a specified number of rows from the beginning (head) or end (tail) of the DataFrame.
- ***It is important to note that this program requires an internet connection in order to access the csv file from the URL.*** If there is no internet connection, the user should create a local csv file with 15 lines and use the following code to read the csv file into a DataFrame:

```
df = pd.read_csv("<local_csv_file_path>")
```

Program 15: **Write a python program which accepts the radius of a circle from user and computes the area (use math module)**

```
import math

# Get the radius of the circle from the user
radius = float(input("Enter the radius of the circle : "))

# Calculate the area of the circle
area = math.pi * radius**2

# Print the result
print("The area of the circle is :", area)
```

Output:

```
Enter the radius of the circle : 5
The area of the circle is : 78.53981633974483
```

Explanation:

- The above program calculates the area of a circle given the radius. The user inputs the radius of the circle, which is then used to calculate the area of the circle using the formula πr^2.
- The math module is imported to use the value of π in the calculation. The user input is received using the input() function and is stored as a string. The string is then converted to a float using the float() function, allowing the radius to be used in mathematical calculations. The area of the circle is calculated and stored in a variable, area. Finally, the value of area is printed to the console.

Program 16: **Use the following data (load it as CSV file) for this exercise. Read this file using Pandas or NumPy or using in-built matplotlib function.**

Months	Pen	Book	Marker	Chair	Table	Pen Stand	Total Units	Total Profit
1.	2500	1500	5200	9200	1200	1500	21100	211000
2.	2630	1200	5100	6100	2100	1200	18330	183300
3.	2140	1340	4550	9550	3550	1340	22470	224700
4.	3400	1130	5870	8870	1870	1130	22270	222700
5.	3600	1740	4560	7760	1560	1740	20960	209600
6.	2760	1555	4890	7490	1890	1555	20140	201400
7.	2980	1120	4780	8980	1780	1120	29550	295500
8.	3700	1400	5860	9960	2860	1400	36140	361400
9.	3540	1780	6100	8100	2100	1780	23400	234000
10.	1990	1890	8300	10300	2300	1890	26670	266700
11.	2340	2100	7300	13300	2400	2100	41280	412800
12.	2900	1760	7400	14400	1800	1760	30020	300200

Open the text editor like Notepad to create the csv file with the above data and store it as **salesData.csv** as shown below.

```
Months,Pen,Book,Marker,Chair,Table,Pen Stand,Total Units,Total Profit
1,2500,1500,5200,9200,1200,1500,21100,211000
2,2630,1200,5100,6100,2100,1200,18330,183300
3,2140,1340,4550,9550,3550,1340,22470,224700
4,3400,1130,5870,8870,1870,1130,22270,222700
5,3600,1740,4560,7760,1560,1740,20960,209600
6,2760,1555,4890,7490,1890,1555,20140,201400
7,2980,1120,4780,8980,1780,1120,29550,295500
8,3700,1400,5860,9960,2860,1400,36140,361400
9,3540,1780,6100,8100,2100,1780,23400,234000
10,1990,1890,8300,10300,2300,1890,26670,266700
11,2340,2100,7300,13300,2400,2100,41280,412800
12,2900,1760,7400,14400,1800,1760,30020,300200
```

Program 16 (a)

Displaying Sales Trend of All Products

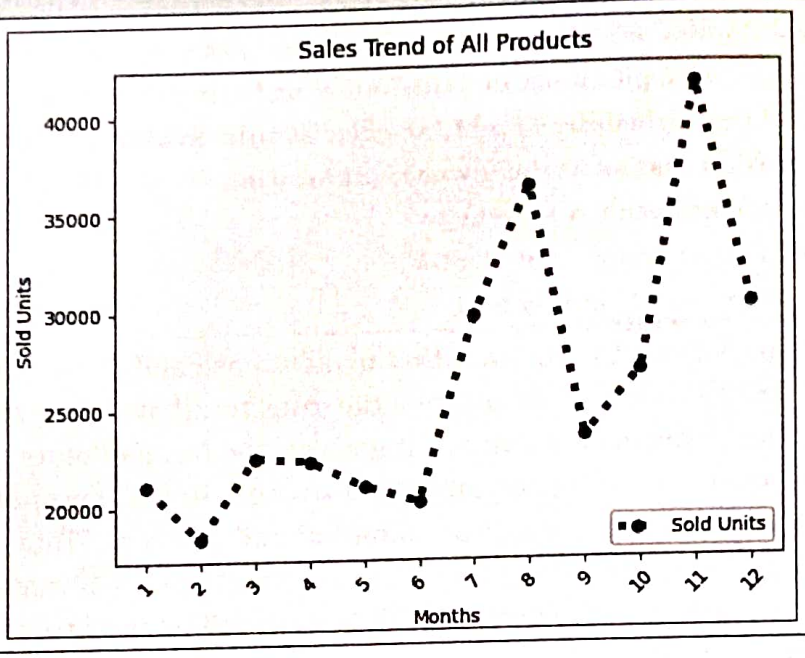
Get total units of all months and show line plot with the following Style properties
Generated line plot must include following Style properties:

- **Line Style dotted and Line-color should be blue**
- **Show legend at the lower right location**
- **X label name = Months**
- **Y label name = Sold Units**
- **Line width should be 4**

Important Note : It's important to plot data that makes sense and is relevant to the analysis we are trying to perform. In the code, we are plotting the sales trend by having the x-axis as "Months" and y-axis as "Total Units". This makes sense as it gives us the trend of units sold over the months. However, marking profits in the y-axis and labeling it as "Sold Units" does not make sense. To plot the profit trend, we need to have the x-axis as "Months" and y-axis as "Total Profit". This gives us the trend of profits earned over the months. So, it's important to plot relevant data in the x-axis and y-axis for the analysis we are trying to perform. We have modified the program title to display the total units of all months in y-axis.

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the csv file into a DataFrame using pd.read_csv()
df = pd.read_csv("salesData.csv")

# Get the data for the "Total Units" column
sold_units = df["Total Units"]

# Plot the total profit data
plt.plot(sold_units, color='blue', marker = 'o', linestyle='dotted', label="Sold
        Units", linewidth=4)

# Add X and Y axis labels
plt.xlabel("Months")
plt.ylabel("Sold Units")

# Add title
plt.title("Sales Trend of All Products")

# Add a legend at the lower right location
plt.legend(loc="lower right")

# Add xticks to include all months
plt.xticks(range(len(df)), df['Months'], rotation=45)

# Show the plot
plt.show()
```

Output:



Explanation:

- The above code is a python program that uses the pandas and matplotlib library to plot a graph. The graph represents the sales trend for a given data.
- The first step in the program is to import the pandas and matplotlib library.
- Next, the program uses pd.read_csv() to read a .csv file called "salesData.csv" and stores the contents of the file into a Pandas DataFrame called df.
- Then, the program gets the data for the "Total Units" column and stores it in the sold_units variable.
- The next step is to plot the data in the sold_units variable. This is done using the plt.plot() function. The function takes several parameters such as the sold_units data, color, marker, linestyle, label, and linewidth. These parameters are used to customize the appearance of the graph.
- After plotting the data, the program adds labels for the x and y-axis using the plt.xlabel() and plt.ylabel() functions. The x-axis represents the months and the y-axis represents the number of units sold.
- The program also adds a title to the graph using the plt.title() function. The title of the graph is "Sales Trend".
- Next, the program adds a legend to the graph using the plt.legend() function. The legend is placed at the lower right location on the graph.
- The program then uses the plt.xticks() function to include all the months on the x-axis. The function takes two parameters, the range of the data and the corresponding values for each tick. In this case, the range of the data is the length of the df DataFrame and the values are the months in the df['Months'] column.
- Finally, the program shows the plot using the plt.show() function. This function displays the graph on the screen.

Program 16 (a)

Displaying Profit Trend of All Products

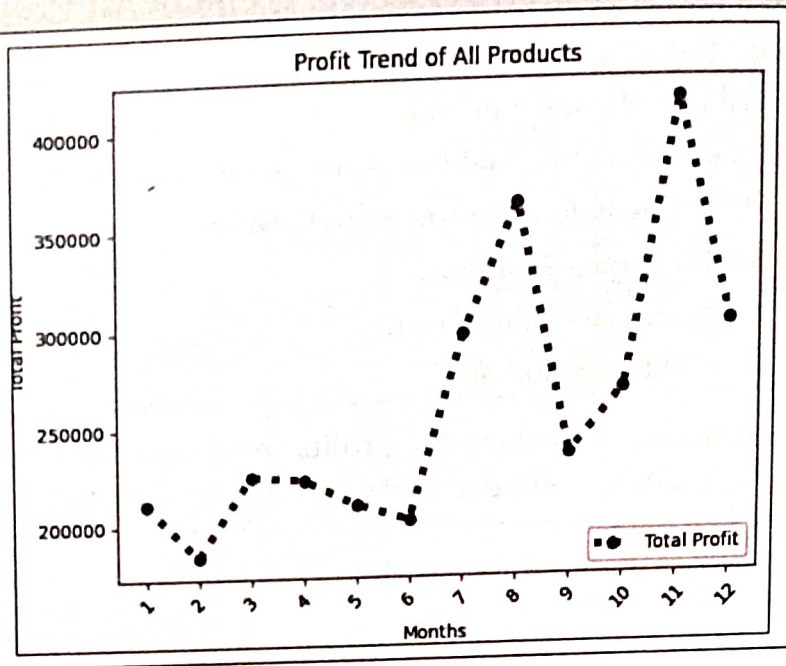
Get total profit of all months and show line plot with the following Style properties Generated line plot must include following Style properties:

- **Line Style dotted and Line-color should be blue**
- **Show legend at the lower right location**
- **X label name = Months**
- **Y label name = Total Profit**
- **Line width should be 4**

Important Note : This program is to show the profits trend over the months. To plot the profit trend, we need to have the x-axis as "Months" and y-axis as "Total Profit".

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the csv file into a DataFrame using pd.read_csv()
df = pd.read_csv("salesData.csv")

# Get the data for the "Total Profit" column
total_profit = df["Total Profit"]

# Plot the total profit data
plt.plot(total_profit, color='blue', marker = 'o', linestyle='dotted', label="Total
        Profit", linewidth=4)

# Add X and Y axis labels
plt.xlabel("Months")
plt.ylabel("Total Profit")

# Add title
plt.title("Profit Trend of All Products")

# Add a legend at the lower right location
plt.legend(loc="lower right")

# Add xticks to include all months
plt.xticks(range(len(df)), df['Months'], rotation=45)

# Show the plot
plt.show()
```

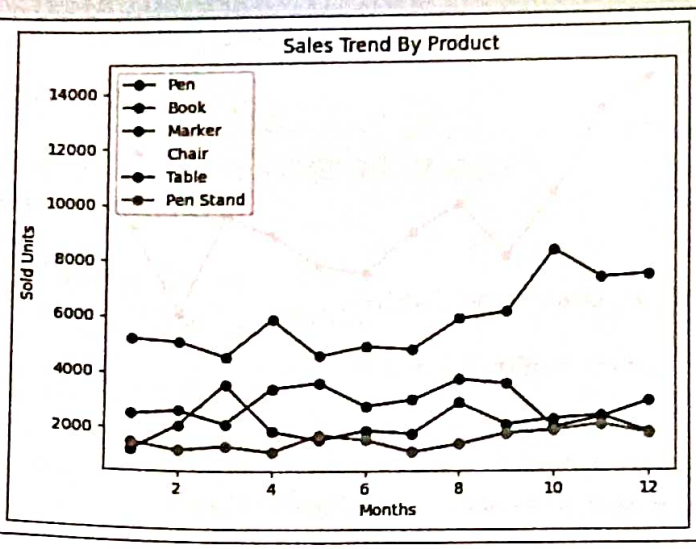
Output:



Program 16 (b) **Display the number of units sold per month for each product using multiline plots. (i.e., Separate Plotline for each product.**

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the csv file into a DataFrame using pd.read_csv()
df = pd.read_csv("salesData.csv")

# Plot the units sold data for each product
plt.plot(df["Months"], df['Pen'], color='blue', marker = 'o', label="Pen")
plt.plot(df["Months"], df['Book'], color='red', marker = 'o', label="Book")
plt.plot(df["Months"], df['Marker'], color='green', marker = 'o', label="Marker")
plt.plot(df["Months"], df['Chair'], color='yellow', marker = 'o', label="Chair")
plt.plot(df["Months"], df['Table'], color='purple', marker = 'o', label="Table")
plt.plot(df["Months"], df['Pen Stand'], color='orange', marker = 'o', label="Pen Stand")

# Add X and Y axis labels
plt.xlabel("Months")
plt.ylabel("Sold Units")

# Add title
plt.title("Sales Trend By Product")

# Add a legend at the upper right location
plt.legend(loc="upper left")

# Show the plot
plt.show()
```

Output:



Explanation:

- The program reads in a csv file called "salesData.csv" using the pandas library and stores the data in a DataFrame. The DataFrame stores the number of units sold of each product for each month. Then, the program plots the sales trend of each product over time by plotting the units sold on the y-axis and the months on the x-axis. The program uses the matplotlib library to plot the data, giving each product a different color, marker and label. Finally, the program adds x and y axis labels, a title, a legend, and displays the plot. The resulting plot provides a visual representation of the sales trend of each product over time, allowing for easy comparison between products.

Note : While observing the output of the plot, it is noticeable that the line for the "Book" product is not shown. This is because the line for the "Pen Stand" product, which has the same sales data as the "Book" product, is plotted last, and therefore it overrides the line for the "Book" product. If the sales data for either the "Book" product or the "Pen Stand" product was different, the line for the "Book" product would also be visible in the plot.

Program 16 (c)

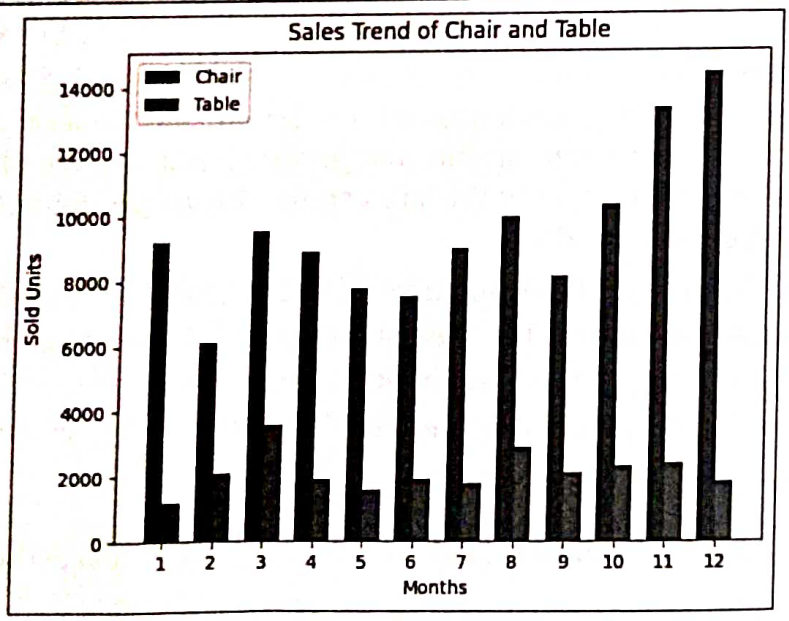
Read chair and table product sales data and show it using the bar chart.

- **The bar chart should display the number of units sold per month for each product. Add a separate bar for each product in the same chart.**

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the csv file into a DataFrame using pd.read_csv()
df = pd.read_csv("salesData.csv")

# Extract the data for Chair and Table products
data = df[["Months", "Chair", "Table"]]

# Create a bar chart for Chair and Table products
bar_width = 0.35
```

```
chair_bar = plt.bar(data.index, data['Chair'], bar_width, color='blue', label='Chair')
table_bar = plt.bar(data.index + bar_width, data['Table'], bar_width, color='red',
label='Table')

# Add X and Y axis labels
plt.xlabel("Months")
plt.ylabel("Sold Units")

# Add title
plt.title("Sales Trend of Chair and Table")

# Add a legend at the upper right location
plt.legend(loc="upper left")

# Set the X-axis tick labels to be the months
plt.xticks(data.index + bar_width/2, data['Months'])

# Show the plot
plt.show()
```

Output:



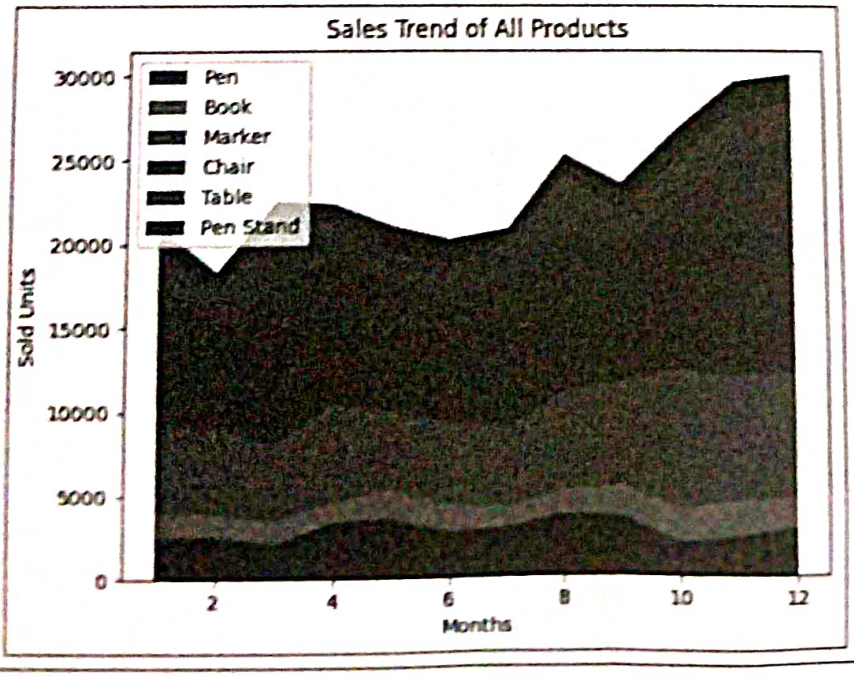
Explanation:

- This code imports the pandas and matplotlib.pyplot libraries in Python. It then uses pandas to read a csv file "salesData.csv" into a DataFrame named df. The code extracts the data for Chair and Table products and stores it in the data variable.
- Next, it creates a bar chart using matplotlib.pyplot library to visualize the sales trend of Chair and Table products. It creates two bars, one for Chair products and the other for Table products. The X-axis displays the months, the Y-axis displays the number of sold units. The chart has a title "Sales Trend of Chair and Table", and a legend located at the upper left corner. The X-axis tick labels are set to be the months, and the chart is finally displayed using plt.show().

Program 16 (d) **Read all product sales data and show it using the stack plot.**

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the csv file into a DataFrame using pd.read_csv()
df = pd.read_csv("salesData.csv")

# Extract the data for all products
data = df[["Months", "Pen", "Book", "Marker", "Chair", "Table", "Pen Stand"]]

# Plot the data using stack plot
plt.stackplot(data["Months"], data["Pen"], data["Book"], data["Marker"],
              data["Chair"], data["Table"], data["Pen Stand"],
              labels=["Pen", "Book", "Marker", "Chair", "Table", "Pen Stand"])

# Add X and Y axis labels
plt.xlabel("Months")
plt.ylabel("Sold Units")

# Add a legend at the upper right location
plt.legend(loc="upper left")

# Add title
plt.title("Sales Trend of All Products")

# Show the plot
plt.show()
```

Output:



Explanation:

- The code imports the pandas and matplotlib.pyplot libraries in Python. It then uses pandas to read a csv file "salesData.csv" into a DataFrame named df. The code extracts the data for all products and stores it in the data variable.
- Next, it creates a stack plot using matplotlib.pyplot library to visualize the sales trend of all products (Pen, Book, Marker, Chair, Table, Pen Stand). The X-axis displays the months, and the Y-axis displays the number of sold units. The stack plot shows the cumulative sum of sales for each product, making it easy to see the relative proportions of sales for each product.
- The chart has a title "Sales Trend of All Products" and a legend located at the upper left corner. The X-axis displays the months and the Y-axis displays the number of sold units. The chart is finally displayed using plt.show().