Sample APL programs

1. Program to check if a number is odd or even in python.

```
# Program to check if a number is odd or even
# Function to check odd or even
def check_odd_even(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
# Input from user
num = int(input("Enter a number: "))
# Check and display the result
result = check_odd_even(num)
print(f"The number {num} is {result}.")
```

2. Find the factorial of a number in python.

```
# Function to calculate factorial
def calculate_factorial(n):
    factorial = 1
    for i in range(1, n + 1):
        factorial *= i
    return factorial
# Input from user
num = int(input("Enter a number: "))
# Check for negative input
if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    # Calculate and display the factorial
    result = calculate_factorial(num)
    print(f"The factorial of {num} is {result}.")
```

3. Find factorial of a number using recursion in python.

```
# Function to calculate factorial using recursion
def calculate_factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * calculate_factorial_recursive(n - 1)

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

# Check for negative input
if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    # Calculate and display the factorial
    result = calculate_factorial_recursive(num)
    print(f"The factorial of {num} is {result}.")
```

4. program to read the contents from a file and display on console
5. program to read the contents from a one file and write into another file
6. program to display file available in current directory
7. Python program to display a horizontal bar chart of the popularity of programming Languages.

   Sample data:

   Programming languages: Java, Python, PHP, JavaScript, C#, C++

   Popularity: 22.2, 17.6, 8.8, 8, 7.7, 6.7

```
import matplotlib.pyplot as plt

# Sample data
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]

# Create a horizontal bar chart
plt.barh(languages, popularity, color='skyblue')
```

```
# Adding labels and title
plt.xlabel('Popularity (%)')
plt.title('Popularity of Programming Languages')

# Show the chart
plt.show()
pip install matplotlib
```

8. Python program to create a pie chart.

```
import matplotlib.pyplot as plt
# Sample data
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
# Create a pie chart
plt.pie(popularity, labels=languages, autopct='%1.1f%%', startangle=140)
# Adding a title
plt.title('Popularity of Programming Languages')
# Show the chart
plt.show()
```

9. program for designing a GUI using Tkinter

```
import tkinter as tk
def on_button_click():
    label.config(text="Button Clicked!")
# Create the main window
window = tk.Tk()
window.title("Simple GUI Example")
# Create a label
label = tk.Label(window, text="Welcome to Tkinter GUI")
label.pack(pady=10)
# Create a button
button = tk.Button(window, text="Click me!", command=on_button_click)
button.pack(pady=10)
```

```
# Run the Tkinter event loop
window.mainloop()
```

10. program for class, object and constructor using suitable data

```
class Student:
    # Constructor
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.grade = grade

    # Method to display student information
    def display_info(self):
        print("Student Name: " + self.name)
        print("Age: " + str(self.age))  # Note: age is an integer, so we convert it to a string before concatenation
        print("Grade: " + self.grade)

# Creating an object of the Student class using the constructor
student1 = Student("John Doe", 18, "A")

# Accessing object attributes using string concatenation
print("Accessing Object Attributes:")
print("Name: " + student1.name)
print("Age: " + str(student1.age))  # Note: age is an integer, so we convert it to a string before concatenation
print("Grade: " + student1.grade)

# Using a method of the class
print("\nUsing Class Method:")
student1.display_info()
```

11. program for Method overloading

```
class MathOperations:
    # Method to add numbers with default values
```

```
    def add(self, a, b=0, c=0):
        return a + b + c

# Create an instance of the MathOperations class
math_ops = MathOperations()

# Test method overloading
result1 = math_ops.add(5)
result2 = math_ops.add(5, 3)
result3 = math_ops.add(5, 3, 2)

# Display results
print("Result 1:", result1)
print("Result 2:", result2)
print("Result 3:", result3)
```

12. program for Method overriding

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    # Overriding the speak method in the subclass
    def speak(self):
        print("Dog barks")

class Cat(Animal):
    # Overriding the speak method in the subclass
    def speak(self):
        print("Cat meows")

# Create instances of the subclasses
dog = Dog()
cat = Cat()
```

```
# Demonstrate method overriding
print("Dog:")
dog.speak()  # This will call the overridden method in the Dog class

print("\nCat:")
cat.speak()  # This will call the overridden method in the Cat class
```

13. program for Single inheritance

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

    def speak(self):
        print(f"{self.name} makes a sound")

# Derived class inheriting from Animal
class Dog(Animal):
    def bark(self):
        print(f"{self.name} barks")

# Create an instance of the derived class
my_dog = Dog("Buddy")

# Access methods from the base class
my_dog.speak()  # Calls the speak method from the Animal class

# Access methods from the derived class
my_dog.bark()  # Calls the bark method from the Dog class
```

14. program for multilevel inheritance

```
# Base class
class Animal:
    def __init__(self, name):
```

```
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound")

# Intermediate class inheriting from Animal
class Mammal(Animal):
    def give_birth(self):
        print(f"{self.name} gives birth to live young")

# Derived class inheriting from Mammal
class Dog(Mammal):
    def bark(self):
        print(f"{self.name} barks")

# Create an instance of the derived class
my_dog = Dog("Buddy")

# Access methods from the base class
my_dog.speak()  # Calls the speak method from the Animal class

# Access methods from the intermediate class
my_dog.give_birth()  # Calls the give_birth method from the Mammal class

# Access methods from the derived class
my_dog.bark()  # Calls the bark method from the Dog class
```

15. program for demonstrating any 5 functions of Array

```
import numpy as np

# Creating an array
arr = np.array([1, 2, 3, 4, 5])

# Function 1: Accessing elements
print("Array:", arr)
```

```
print("Element at index 2:", arr[2])

# Function 2: Getting array shape
print("Shape of array:", arr.shape)

# Function 3: Reshaping the array
arr_reshaped = arr.reshape(1, 5)
print("Reshaped array:", arr_reshaped)

# Function 4: Sum of array elements
print("Sum of array elements:", np.sum(arr))

# Function 5: Finding the maximum element
print("Maximum element:", np.max(arr))
```

16. program for demonstrating any 5 functions of list

```
# Creating a list
my_list = [1, 2, 3, 4, 5]

# Function 1: Accessing elements
print("List:", my_list)
print("Element at index 2:", my_list[2])

# Function 2: Appending an element
my_list.append(6)
print("List after appending:", my_list)

# Function 3: Removing an element
my_list.remove(3)
print("List after removing:", my_list)

# Function 4: Checking if an element exists
print("Does 4 exist in the list?", 4 in my_list)
```

```
# Function 5: Finding the index of an element
index_of_5 = my_list.index(5)
print("Index of 5:", index_of_5)
```

17. program for demonstrating any 5 functions of dictionary

```
# Creating a dictionary
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

# Function 1: Accessing values
print("Dictionary:", my_dict)
print("Age:", my_dict['age'])

# Function 2: Adding a new key-value pair
my_dict['occupation'] = 'Engineer'
print("Dictionary after adding a key-value pair:", my_dict)

# Function 3: Removing a key-value pair
my_dict.pop('city')
print("Dictionary after removing 'city':", my_dict)

# Function 4: Checking if a key exists
print("Does 'gender' exist in the dictionary?", 'gender' in my_dict)

# Function 5: Getting all keys
keys = my_dict.keys()
print("Keys in the dictionary:", keys)
```

18. program for demonstrating any 5 functions of string

```
# Creating a string
my_string = "Hello, World!"

# Function 1: Length of the string
print("Length of the string:", len(my_string))
```

```
# Function 2: Converting to uppercase
uppercase_string = my_string.upper()
print("Uppercase string:", uppercase_string)

# Function 3: Finding a substring
index_of_world = my_string.find('World')
print("Index of 'World':", index_of_world)

# Function 4: Splitting the string
split_string = my_string.split(',')
print("Split string:", split_string)

# Function 5: Replacing a substring
new_string = my_string.replace('World', 'Universe')
print("String after replacement:", new_string)
```

19. program for demonstrating any 5 functions of tuple

```
# Creating a tuple
my_tuple = (1, 2, 3, 4, 5)

# Function 1: Accessing elements
print("Tuple:", my_tuple)
print("Element at index 2:", my_tuple[2])

# Function 2: Concatenating tuples
new_tuple = my_tuple + (6, 7)
print("Concatenated tuple:", new_tuple)

# Function 3: Count occurrences of an element
count_of_4 = my_tuple.count(4)
print("Count of 4:", count_of_4)

# Function 4: Finding the index of an element
index_of_3 = my_tuple.index(3)
```

```
print("Index of 3:", index_of_3)

# Function 5: Getting the length of the tuple
length_of_tuple = len(my_tuple)
print("Length of the tuple:", length_of_tuple)
```

20. program for demonstrating any 5 functions of set

```
# Creating a set
my_set = {1, 2, 3, 4, 5}

# Function 1: Adding an element
my_set.add(6)
print("Set after adding an element:", my_set)

# Function 2: Removing an element
my_set.remove(3)
print("Set after removing an element:", my_set)

# Function 3: Checking if an element exists
print("Does 4 exist in the set?", 4 in my_set)

# Function 4: Union of two sets
another_set = {5, 6, 7, 8, 9}
union_set = my_set.union(another_set)
print("Union of two sets:", union_set)

# Function 5: Clearing all elements in the set
my_set.clear()
print("Set after clearing all elements:", my_set)
```

21. program for creating function and lambda function

```
# Regular function
def add_numbers(x, y):
    return x + y
```

```
# Lambda function (anonymous function)
multiply = lambda x, y: x * y

# Test regular function
result_add = add_numbers(3, 4)
print("Result of add_numbers function:", result_add)

# Test lambda function
result_multiply = multiply(3, 4)
print("Result of lambda function (multiply):", result_multiply)
```

22. program for Creating a new package for addition and multiplication operation and use it in another program.

```
math_operations/
|-- __init__.py
|-- addition.py
|-- multiplication.py
main_program.py

# This file is needed to treat the 'math_operations' folder as a Python package

def add(x, y):
    return x + y

def multiply(x, y):
    return x * y

# Importing functions from the 'math_operations' package
from math_operations import addition, multiplication

# Using the functions from the package
result_add = addition.add(3, 4)
result_multiply = multiplication.multiply(3, 4)
```

```
# Displaying results
print("Result of addition:", result_add)
print("Result of multiplication:", result_multiply)
```

23. program for use of Numpy library

```
import numpy as np

# Creating NumPy arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([6, 7, 8, 9, 10])

# Element-wise addition
result_addition = np.add(arr1, arr2)
print("Element-wise Addition:", result_addition)

# Element-wise multiplication
result_multiplication = np.multiply(arr1, arr2)
print("Element-wise Multiplication:", result_multiplication)

# Dot product
dot_product = np.dot(arr1, arr2)
print("Dot Product:", dot_product)

# Matrix multiplication
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
result_matrix_multiplication = np.matmul(matrix1, matrix2)
print("Matrix Multiplication:")
print(result_matrix_multiplication)

# Reshaping arrays
arr3 = np.arange(12)
reshaped_arr = arr3.reshape(3, 4)
```

```
print("Original Array:")
print(arr3)
print("Reshaped Array:")
print(reshaped_arr)

# Statistical operations
mean_value = np.mean(arr1)
std_deviation = np.std(arr2)
print("Mean Value:", mean_value)
print("Standard Deviation:", std_deviation)

pip install numpy
```

24. program for use of pandas library

```
import pandas as pd


# Creating a DataFrame
data = {
    'Name': ['John', 'Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 28, 22, 30, 35],
    'City': ['New York', 'San Francisco', 'Chicago', 'Los Angeles', 'Seattle']
}

df = pd.DataFrame(data)

# Displaying the DataFrame
print("Original DataFrame:")
```

```
print(df)

# Accessing columns
ages = df['Age']
print("\nAges Column:")
print(ages)

# Adding a new column
df['Occupation'] = ['Engineer', 'Designer', 'Teacher', 'Actor', 'Doctor']
print("\nDataFrame with Occupation Column:")
print(df)

# Filtering data
young_people = df[df['Age'] < 30]
print("\nPeople younger than 30:")
print(young_people)

# Statistical summary
summary = df.describe()
print("\nStatistical Summary:")
print(summary)

# Saving DataFrame to a CSV file
```

```
df.to_csv('people.csv', index=False)


# Reading data from a CSV file

read_df = pd.read_csv('people.csv')

print("\nDataFrame read from CSV:")

print(read_df)


pip install pandas
```