Week 3; Python programs based on lists, dictionaries, tuples, strings, regular expressions, and OOP concepts:

1. Lists

Program:Reverse a list

```
def reverse_list(lst):
   return lst[::-1]


sample_list = [1, 2, 3, 4, 5]
print("Original list:", sample_list)
print("Reversed list:",
reverse_list(sample_list))
```

2. Dictionaries

Program: Count the frequency of elements in a list using a dictionary

```
def count_frequency(lst):
    freq_dict = {}
    for item in lst:
        if item in freq_dict:
            freq_dict[item] += 1
        else:
```

```
        freq_dict[item] = 1
    return freq_dict


sample_list = ['apple', 'banana',
'apple', 'orange', 'banana', 'apple']
print("Frequency count:",
count_frequency(sample_list))
```

3. Tuples

Program: Find the maximum and minimum elements in a tuple

```
def max_min_tuple(tpl):
    return max(tpl), min(tpl)


sample_tuple = (4, 7, 1, 3, 9, 2)
max_val, min_val = max_min_tuple(sample_tuple)
print("Maximum value:", max_val)
print("Minimum value:", min_val)
```

4. Strings

Program: Check if a string is a palindrome

```
def is_palindrome(s):
    return s == s[::-1]


sample_string = "radar"
print(f"Is '{sample_string}' a palindrome?", is_palindrome(sample_string))
```

5. Regular Expressions

Program: Extract all email addresses from a text

```
import re
```

```
def extract_emails(text):
   email_pattern = r'[a-zA-Z0-
9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]
{2,}'
   return re.findall(email_pattern,
text)


sample_text = "Please contact us at
support@example.com or
sales@example.org."
emails =
extract_emails(sample_text)
print("Extracted emails:", emails)
```

6. OOP Concepts - Classes and Objects

rogram: Define a class `Person` with attributes and methods

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

   def greet(self):
      print(f"Hello, my name is
{self.name} and I am {self.age} years
old.")
```

```
person1 = Person("Alice", 30)
person1.greet()
```

7. OOP Concepts - Inheritance

Program:Define a subclass `Student` that inherits from `Person`

```
class Student(Person):
  def __init__(self, name, age, student_id):
    super().__init__(name, age)
    self.student_id = student_id
    def display_id(self):
        print(f"My student ID is {self.student_id}.")

student1 = Student("Bob", 20, "S12345")
student1.greet()
student1.display_id()
```

8. OOP Concepts - Encapsulation

Program:Demonstrate encapsulation with private

attributes

```
class BankAccount:
    def __init__(self, account_number,
balance):
        self.__account_number =
account_number
        self.__balance = balance

    def deposit(self, amount):
        self.__balance += amount
        print(f"Deposited {amount}. New
balance is {self.__balance}.")
```

```
    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrew {amount}.
New balance is {self.__balance}.")
        else:
            print("Insufficient balance.")


account =
BankAccount("123456789", 1000)
account.deposit(500)
account.withdraw(200)
```

9. OOP Concepts - Polymorphism

Program:Demonstrate method overriding

```
class Animal:
    def sound(self):
        print("Some generic animal
sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

class Cat(Animal):
    def sound(self):
        print("Meow")

animals = [Animal(), Dog(), Cat()]
for animal in animals:
    animal.sound()
```

10. OOP Concepts - Abstraction

Program: Implement an abstract class

```
from abc import ABC,
abstractmethod
```

```
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass


class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height


    def area(self):
        return self.width * self.height
```

```
rectangle = Rectangle(10, 5)
print("Rectangle area:",
rectangle.area())
```