ASSIGNMENT 2

<1.>Write a program to count word frequencies in a given text:-

Code :-

```
import re

def word_frequencies_v2(text):
    # convert all into the lowercase
    text = text.casefold()

    # remove punctuation and add only words
    words = re.findall(r"[A-Za-z0-9']+", text)

    # make empty dictionary to store frequency
    freq = {}
    # increase the count of the words manually
    for word in words:
        if word in freq:
            freq[word] += 1
        else:
            freq[word] = 1

    return freq

# Example run
sample_text = "Hello, I am vishal mohil , hello! This is a test. This test is simple, simple."
print(word_frequencies_v2(sample_text))
```

output :-

```
{'hello': 2, 'i': 1, 'am': 1, 'vishal': 1, 'mohil': 1, 'this': 2, 'is': 2, 'a': 1, 'test': 2, 'simple': 2}
```

<2.> Palimdrome Checker – Write a program that checks if a given word is a palindrome:-

Code :-

```
def is_palindrome(word):
    # keep only alphanumeric character and convert them into the lowecase
    cleaned = ""
    for i in word:
        if i.isalnum():
            cleaned += i.lower()
            # make reverse string and then compare
    return cleaned == cleaned[::-1]
print(is_palindrome("mam"))
print(is_palindrome("python"))
print(is_palindrome("non"))
```

output :-

```
True

False

True
```

<3.> List Manipulation – Create a list of numbers , then write a program that prints the square of each number in the list :-

Code :-

```
def squares(numbers):
    return list(map(lambda x: x * x, numbers))

# for Example
nums = [1, 2, 3, 4, 5, 6]
print("Numbers:", nums)
print("Squares:", squares(nums))
```

output :-

```
Numbers: [1, 2, 3, 4, 5, 6]

Squares: [1, 4, 9, 16, 25, 36]
```

Object-Oriented Programming (OOP) in Python

By :- VISHAL MOHIL

What is OOP

- A programming paradigm based on objectsObjects = combination of data (attributes) and methods (functions)

- Provides structure for reusability and scalability
- Python fully supports OOP principles

Key OOP Concepts

- Class ‘n - Blueprint for creating objects
- Object’n - Instance of a class
- Inheritance ‘n - Acquire properties from another class

- Polymorphism ‘n - Many forms, same interface
- Encapsulation ‘n- Restrict access to data
- Abstraction ‘n Hiding implementation details

Classes and Objects example:-

- class Car:
- def _init_(self, brand, model):

- self.brand = brand
- self.model = modelmy_car = Car("Toyota", "Corolla")
- print(my_car.brand, my_car.model)
- Output: Toyota Corolla

Inheritance Example:-

- class Animal:
- def speak(self):

- print("Animal speaks")
- class Dog(Animal):
- def speak(self):
- print("Dog barks")
- dog = Dog()dog.speak()
- Output: Dog barks

Polymorphisn Example :-

class Bird:

```
   def fly(self):
print("Flying in the sky")
class Airplane:     def
fly(self):
  print("Flying using fuel") for
obj in [Bird(), Airplane()]:
  obj.fly()
```

- Shows same method (fly) behaving differently

Encapsulation and Abstraction :-

- Encapsulation:
- Wrapping data & methods together
- Example: Private variables with __name
- Abstraction:Hiding implementation details
- Example: Abstract Base Classes (abc module)

Advantages of OOP :-

- Code reusability

- Modular and easy to maintain
- Easier to debug and update
- Models real-world problems effectively
- Improves scalability of projects