1. Define a Python class named Car with attributes such as make, model, and year. Create two objects of the class and print their details.

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

def display(self):

print(f"{self.year} {self.make} {self.model}")

car1 = Car("Toyota", "Corolla", 2020)

car2 = Car("Honda", "Civic", 2022)

car1.display()

car2.display()

2. Implement a Python class named Person with private attributes for name, age, and address. Provide methods to get and set these attributes. Demonstrate encapsulation by accessing these attributes only through the methods.

class Person:

def \_\_init\_\_(self, name, age, address):

self.\_\_name = name

self.\_\_age = age

self.\_\_address = address

def get\_name(self):

return self.\_\_name

def set\_name(self, name):

self.\_\_name = name

def get\_age(self):

return self.\_\_age

def set\_age(self, age):

self.\_\_age = age

def get\_address(self):

return self.\_\_address

def set\_address(self, address):

self.\_\_address = address

# Demonstration

p = Person("John", 30, "New York")

print(p.get\_name(), p.get\_age(), p.get\_address())

p.set\_name("Alice")

p.set\_age(25)

p.set\_address("Los Angeles")

print(p.get\_name(), p.get\_age(), p.get\_address())

3. Create an abstract class Shape with an abstract method area. Implement two concrete classes, Circle and Rectangle, that inherit from Shape and provide their own implementations of the area method. Demonstrate polymorphism by calling the area method on objects of both classes.

from abc import ABC, abstractmethod

import math

class Shape(ABC):

@abstractmethod

def area(self):

pass

class Circle(Shape):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return math.pi \* self.radius \*\* 2

class Rectangle(Shape):

def \_\_init\_\_(self, length, width):

self.length = length

self.width = width

def area(self):

return self.length \* self.width

# Polymorphism

shapes = [Circle(5), Rectangle(4, 6)]

for shape in shapes:

print("Area:", shape.area())

4. Define a base class called Animal with a method speak. Create two derived classes, Dog and Cat, which inherit from the Animal class. Override the speak method in each derived class to make dogs bark and cats meow.

class Animal:

def speak(self):

print("Animal speaks")

class Dog(Animal):

def speak(self):

print("Dog barks")

class Cat(Animal):

def speak(self):

print("Cat meows")

dog = Dog()

cat = Cat()

dog.speak()

cat.speak()

5. Create a class with a static variable that counts the number of instances created. Demonstrate how the static variable is shared among all instances.

class InstanceCounter:

count = 0 # static variable

def \_\_init\_\_(self):

InstanceCounter.count += 1

obj1 = InstanceCounter()

obj2 = InstanceCounter()

obj3 = InstanceCounter()

print("Number of instances created:", InstanceCounter.count)

6. Explain the use of the final keyword in Python. Provide an example scenario where using final would be appropriate.

Answer:

Python provides final from the typing module to prevent variables or methods from being overridden.

from typing import Final

PI: Final = 3.14159 # Constant value

def area\_circle(radius):

return PI \* radius \* radius

print(area\_circle(5))

7. Develop a Python class with private, protected, and public attributes. Create methods to access and modify these attributes, demonstrating the impact of different access modifiers.

class Demo:

def \_\_init\_\_(self):

self.public\_var = "Public"

self.\_protected\_var = "Protected"

self.\_\_private\_var = "Private"

def get\_private(self):

return self.\_\_private\_var

def set\_private(self, value):

self.\_\_private\_var = value

obj = Demo()

print("Public:", obj.public\_var)

print("Protected:", obj.\_protected\_var)

print("Private (accessed via method):", obj.get\_private())

8. Define a class with a class variable. Demonstrate how the scope of the class variable differs from the instance variable and create instances to illustrate this scope.

class Example:

class\_var = "I am a class variable"

def \_\_init\_\_(self, instance\_var):

self.instance\_var = instance\_var

obj1 = Example("Instance 1")

obj2 = Example("Instance 2")

print(obj1.class\_var, "|", obj1.instance\_var)

print(obj2.class\_var, "|", obj2.instance\_var)

Example.class\_var = "Updated class variable"

print(obj1.class\_var)

print(obj2.class\_var)

9. Explain the concept of file handling in Python. Mention different file modes and their purposes.

Answer:

File handling allows reading from and writing to files. Common file modes:

* 'r': Read mode
* 'w': Write mode (overwrites existing file)
* 'a': Append mode
* 'b': Binary mode
* 'x': Create a new file

with open("sample.txt", "w") as f:

f.write("Hello, world!")

10. Write a Python script that reads a text file, counts the occurrences of each word, and writes the word frequencies to a new file. Handle potential exceptions during file operations.

from collections import Counter

try:

with open("input.txt", "r") as file:

text = file.read().lower()

words = text.split()

word\_count = Counter(words)

with open("output.txt", "w") as out\_file:

for word, count in word\_count.items():

out\_file.write(f"{word}: {count}\n")

except FileNotFoundError:

print("The file does not exist.")

except Exception as e:

print("An error occurred:", e)

11. A given file data.txt has numbers in it. Read the file and print all the prime numbers from the file.

def is\_prime(num):

if num <= 1:

return False

for i in range(2, int(num\*\*0.5)+1):

if num % i == 0:

return False

return True

try:

with open("data.txt", "r") as file:

numbers = map(int, file.read().split())

primes = [str(num) for num in numbers if is\_prime(num)]

print("Prime numbers:", " ".join(primes))

except FileNotFoundError:

print("data.txt not found.")

12. From a given file data.txt, count the number of words, sentences and number of lines and print.

try:

with open("data.txt", "r") as file:

lines = file.readlines()

line\_count = len(lines)

text = "".join(lines)

word\_count = len(text.split())

sentence\_count = text.count('.') + text.count('!') + text.count('?')

print("Lines:", line\_count)

print("Words:", word\_count)

print("Sentences:", sentence\_count)

except FileNotFoundError:

print("data.txt not found.")