**Question 1**

**class** Vehicle:

**def** \_\_init\_\_(self, name):

*# Storing the name of the vehicle*

self.name = name

**def** sound(self):

*# This method should be implemented by subclasses*

**class** **Car** (Vehicle):

**def** sound(self):

**class** **Bike** (Vehicle):

**def** sound(self):

*# Dog-specific sound*

**return** "PEEP"

*# Creating instances*

*# Vehicle instance with generic name*

a = Vehicle("Mazda")

*# Bike instance with name 'Suzuki'*

d = Bike("Suzuki")

*# Accessing attributes and methods*

print(a.name) *# Output: Mazda*

print(d.name) *# Output: Suzuki*

print(d.sound()) *# Output: PEEP!*

**Question 2**

# define a function for calculating

# the area of a shapes

def calculate\_area(name):\

  # converting all characters

  # into lower cases

  name = name.lower()

  # check for the conditions

  if name == "rectangle":

    l = int(input("Enter rectangle's length: "))

    b = int(input("Enter rectangle's breadth: "))

    # calculate area of rectangle

    rect\_area = l \* b

    print(f"The area of rectangle is

          {rect\_area}.")

  elif name == "square":

    s = int(input("Enter square's side length: "))

    # calculate area of square

    sqt\_area = s \* s

    print(f"The area of square is

          {sqt\_area}.")

  elif name == "triangle":

    h = int(input("Enter triangle's height length: "))

    b = int(input("Enter triangle's breadth length: "))

    # calculate area of triangle

    tri\_area = 0.5 \* b \* h

    print(f"The area of triangle is

          {tri\_area}.")

  elif name == "circle":

    r = int(input("Enter circle's radius length: "))

    pi = 3.14

    # calculate area of circle

    circ\_area = pi \* r \* r

    print(f"The area of circle is

          {circ\_area}.")

  elif name == 'parallelogram':

    b = int(input("Enter parallelogram's base length: "))

    h = int(input("Enter parallelogram's height length: "))

    # calculate area of parallelogram

    para\_area = b \* h

    print ("The area of parallelogram is

          {para\_area}.")

  else:

    print ("Sorry! This shape is not available")

# driver code

if \_\_name\_\_ == "\_\_main\_\_" :

  print("Calculate Shape Area")

  shape\_name = input("Enter the name of shape whose area you want to find: ")

  # function calling

  calculate\_area(shape\_name)

**Question 3**

class Shape:  
def \_\_init\_\_(self):  
# Initialization logic in the Shape class  
print("Initializing Shape")  
  
def calculate\_area(self):  
# Base class area method does nothing  
pass  
  
class Rectangle(Shape):  
def \_\_init\_\_(self, width, height):  
# Call the Shape class's constructor using super()  
super().\_\_init\_\_()  
self.height = height

self.width = width  
  
  
def calculate\_area(self):  
# Utilize the initialization logic from the Shape class  
# via the super() call in the Rectangle class's constructor  
area = self.width \* self.height  
return area

**Question 4**

def process\_sound(sound\_object):  
  
Process a sound object by calling its make\_sound method.  
  
Args:  
sound\_object: An object with a make\_sound method.  
  
# Call the make\_sound method on the sound object  
sound\_object.make\_sound()  
  
# Define the Dog class  
class Dog:  
def make\_sound(self):  
# Implementation of the make\_sound method for Dog  
print("Hoof!")  
  
# Define the Cat class  
class Cat:  
def make\_sound(self):  
# Implementation of the make\_sound method for Cat  
print("Meow!")  
  
# Example usage:  
dog = Dog()  
cat = Cat()  
  
# Pass Dog and Cat objects to the process\_sound function  
process\_sound(dog) # Output: Hoof!  
process\_sound(cat) # Output: Meow!

**Question 5**

class FileHandler(ABC):  
@abstractmethod  
def read(self):  
  
Read the contents of the file.  
  
Returns:  
The contents of the file.

pass  
  
@abstractmethod  
def write(self, contents):  
"""  
Write contents to the file.  
  
Args:  
contents: The contents to write to the file.  
"""  
pass  
  
# Define the concrete class TextFileHandler  
class TextFileHandler(FileHandler):  
def \_\_init\_\_(self, filename):  
self.filename = filename  
  
def read(self):  
try:  
with open(self.filename, 'r') as file:  
return file.read()  
except FileNotFoundError:  
return None  
  
def write(self, contents):  
with open(self.filename, 'w') as file:  
file.write(contents)  
  
# Define the concrete class BinaryFileHandler  
class BinaryFileHandler(FileHandler):  
def \_\_init\_\_(self, filename):  
self.filename = filename  
  
def read(self):  
try:  
with open(self.filename, 'rb') as file:  
return file.read()  
except FileNotFoundError:  
return None  
  
def write(self, contents):  
with open(self.filename, 'wb') as file:  
file.write(contents)  
  
# Example usage:  
text\_handler = TextFileHandler('example.txt')  
text\_handler.write('Hello, World!')  
print(text\_handler.read()) # Output: Hello, World!  
  
binary\_handler = BinaryFileHandler('example.bin')  
binary\_handler.write(b'Hello, World!')  
print(binary\_handler.read()) # Output: b'Hello, World!'  
```