# Python Abstract Classes - Practical Questions

## Question 1: Basic Abstract Class Implementation

Create an abstract class `Animal` with an abstract method `make\_sound()`. Then, implement two child classes:  
- `Dog` which prints "Bark!"  
- `Cat` which prints "Meow!"  
Create objects of both child classes and call the `make\_sound()` method.

## Question 2: Abstract Class with Constructor

Create an abstract class `Vehicle` with:  
- An abstract method `max\_speed()`  
- A constructor that initializes `brand` of the vehicle  
Implement two child classes:  
- `Car`, where `max\_speed()` returns 200 km/h  
- `Bike`, where `max\_speed()` returns 150 km/h  
Print the brand and maximum speed for both objects.

## Question 3: Using Abstract Methods in Derived Classes

Create an abstract class `Shape` with an abstract method `area()`. Implement two subclasses:  
- `Rectangle` with `area()` returning \*\*length × breadth\*\*  
- `Circle` with `area()` returning \*\*π × radius²\*\*  
Instantiate both classes and print their respective areas.

## Question 4: Calling Abstract Methods from Parent Class

Modify the `Father` class to include:  
- An abstract method `profession()`  
- A method `introduce()` that calls `profession()` inside it  
Then, create two child classes (`Engineer`, `Doctor`) that implement `profession()` and demonstrate calling `introduce()`.

## Question 5: Multiple Inheritance with Abstract Classes

Create two abstract classes:  
- `Person` with an abstract method `get\_name()`  
- `Employee` with an abstract method `get\_salary()`  
Then, create a `Manager` class that inherits both and implements the methods. Create an object and display the name and salary.

# Answers to Python Abstract Classes - Practical Questions

## Answer 1: Basic Abstract Class Implementation

from abc import ABC, abstractmethod  
  
class Animal(ABC):  
 @abstractmethod  
 def make\_sound(self):  
 pass  
  
class Dog(Animal):  
 def make\_sound(self):  
 print("Bark!")  
  
class Cat(Animal):  
 def make\_sound(self):  
 print("Meow!")  
  
d = Dog()  
d.make\_sound() # Output: Bark!  
  
c = Cat()  
c.make\_sound() # Output: Meow!

## Answer 2: Abstract Class with Constructor

from abc import ABC, abstractmethod  
  
class Vehicle(ABC):  
 def \_\_init\_\_(self, brand):  
 self.brand = brand  
  
 @abstractmethod  
 def max\_speed(self):  
 pass  
  
class Car(Vehicle):  
 def max\_speed(self):  
 return 200  
  
class Bike(Vehicle):  
 def max\_speed(self):  
 return 150  
  
car = Car("Tesla")  
bike = Bike("Yamaha")  
  
print(f"{car.brand} Max Speed: {car.max\_speed()} km/h") # Tesla Max Speed: 200 km/h  
print(f"{bike.brand} Max Speed: {bike.max\_speed()} km/h") # Yamaha Max Speed: 150 km/h

## Answer 3: Using Abstract Methods in Derived Classes

from abc import ABC, abstractmethod  
import math  
  
class Shape(ABC):  
 @abstractmethod  
 def area(self):  
 pass  
  
class Rectangle(Shape):  
 def \_\_init\_\_(self, length, breadth):  
 self.length = length  
 self.breadth = breadth  
  
 def area(self):  
 return self.length \* self.breadth  
  
class Circle(Shape):  
 def \_\_init\_\_(self, radius):  
 self.radius = radius  
  
 def area(self):  
 return math.pi \* self.radius \*\* 2  
  
r = Rectangle(5, 10)  
c = Circle(7)  
  
print(f"Rectangle Area: {r.area()}") # Output: 50  
print(f"Circle Area: {c.area()}") # Output: ~153.94

## Answer 4: Calling Abstract Methods from Parent Class

from abc import ABC, abstractmethod  
  
class Father(ABC):  
 @abstractmethod  
 def profession(self):  
 pass  
  
 def introduce(self):  
 print("I am a father.")  
 self.profession()  
  
class Engineer(Father):  
 def profession(self):  
 print("I am an Engineer.")  
  
class Doctor(Father):  
 def profession(self):  
 print("I am a Doctor.")  
  
e = Engineer()  
e.introduce()  
  
d = Doctor()  
d.introduce()

## Answer 5: Multiple Inheritance with Abstract Classes

from abc import ABC, abstractmethod  
  
class Person(ABC):  
 @abstractmethod  
 def get\_name(self):  
 pass  
  
class Employee(ABC):  
 @abstractmethod  
 def get\_salary(self):  
 pass  
  
class Manager(Person, Employee):  
 def \_\_init\_\_(self, name, salary):  
 self.name = name  
 self.salary = salary  
  
 def get\_name(self):  
 return self.name  
  
 def get\_salary(self):  
 return self.salary  
  
m = Manager("Alice", 90000)  
print(f"Manager Name: {m.get\_name()}") # Output: Alice  
print(f"Manager Salary: {m.get\_salary()}") # Output: 90000