1. What is the concept of an abstract superclass?

An abstract class in Python is typically created to declare a set of methods that must be created in any child class built on top of this abstract class. Similarly, an **abstract method is one that doesn't have any implementation**

from Shape import Shape

class Circle(Shape):

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

super().\_\_init\_\_("circle")

def draw(self):

print("Drawing a Circle")

from Shape import Shape

class Triangle(Shape):

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

super().\_\_init\_\_("triangle")

def draw(self):

print("Drawing a Triangle")

from Circle import Circle

from Triangle import Triangle

#create a circle object

circle = Circle()

circle.draw()

#create a triangle object

triangle = Triangle()

triangle.draw()

2. What happens when a class statement's top level contains a basic assignment statement?

That basic assignment statement will be used in all class which inherits that class.

3. Why does a class need to manually call a superclass's \_\_init\_\_ method?

Because one needs to define something which is not defined/done in base class \_\_init\_\_ and the only possibility to obtain that is to put it’s execution in aderived class \_\_init\_\_ function

4. How can you augment, instead of completely replacing, an inherited method?

Message forwarding allow us to augment an inherited method in such a way that it can perform inherited action and some new action

5. How is the local scope of a class different from that of a function?

class Triangle(Shape):

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

super().\_\_init\_\_("triangle")

def draw(self):

print("Drawing a Triangle")

local scope of class is with in class including functions in the class

and local scope of function is with in the function