



**Module Title**

**Principles of Programming**

**Weekly Assignment - Practical 9**

**Year**

**2025**

**Student Name: NIRVIK K.C.**

**UWE ID: 25024649**

**Assignment Submission Date: January 16, 2026**

This assignment consists of the programming questions from practical exercises related to the topics from week 9.

### **Exercise 1: Understanding the basic concept of Inheritance**

When you inherit from an existing class, you can reuse methods and fields of parent class, and further you can add new methods and fields also to your derived class.

1.

Task 1.1. Write the following program, study the code, save, compile, and run it.

`/* file name should be Shape.py*/`

Given code:

```
class Shape:
```

```
    def displayShape(self):
```

```
        print("Shape is displaying")
```

```
class Rectangle(Shape):
```

```
    def displayRectangle(self):
```

```
        print("Rectangle is displaying")
```

```
rec = Rectangle()  
rec.displayShape()  
rec.displayRectangle()
```

If you can successfully run your program, you should see the following output:

```
Shape is displaying  
Rectangle is displaying
```

**Answer:**

**Following code for input:**

```
# Shape.py - Display the required shape  
# Practical 9 - Exercise 1 (Task 1.1)  
# Nirvik K.C.  
  
class Shape:  
  
    def displayShape(self):  
        print("Shape is displaying")
```

```
# Class Rectangle inherits from Shape
```

```
class Rectangle(Shape):
```

```
    def displayRectangle(self):
```

```
        print("Rectangle is displaying")
```

```
# Create objects
```

```
rec = Rectangle()
```

```
# Call methods
```

```
rec.displayShape()
```

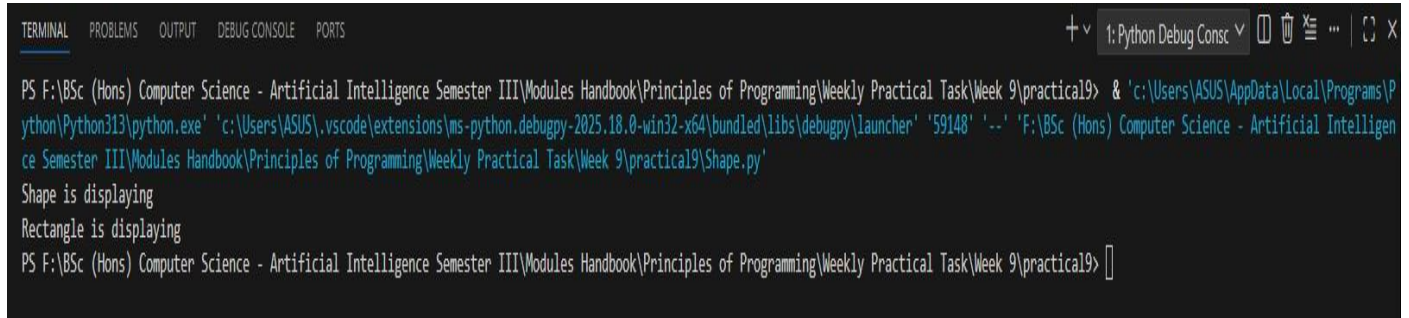
```
rec.displayRectangle()
```

```
# Output:
```

```
# Shape is displaying
```

```
# Rectangle is displaying
```

## Output obtained in execution:



```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59148' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape.py'
Shape is displaying
Rectangle is displaying
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>

```

2.

Task 1.2. You may have noticed that no constructors are specifically given in the above code. Modify the above program by providing the constructors as below, save, compile, and run it.

Given code:

class Shape:

    # constructor for Shape class

    def \_\_init\_\_(self):

        print("Shape constructor is called")

    def displayShape(self):

        print("Shape is displaying")

```
class Rectangle(Shape):  
    # constructor for Rectangle class  
  
    def __init__(self):  
        print("Rectangle constructor is called")  
  
    def displayRectangle(self):  
        print("Rectangle is displaying")  
  
rec = Rectangle()  
rec.displayShape()  
rec.displayRectangle()
```

If you can successfully run your program, you should see the following output:

Rectangle constructor is called

Shape is displaying

Rectangle is displaying

Observation: Unlike other OOP languages, like e.g., C++/Java, Python subclass does not call its base class constructor automatically.

**Answer:**

**Following code for input:**

```
# Showing constructors in inheritance
```

```
# Practical 9 - Exercise 1 (Task 1.2)
```

```
# Nirvik K.C.
```

```
class Shape:
```

```
    # Constructor for Shape class
```

```
    def __init__(self):
```

```
        print("Shape constructor is called")
```

```
    def displayShape(self):
```

```
        print("Shape is displaying")
```

```
class Rectangle(Shape):
```

```
    # Constructor for the class Rectangle
```

```
    def __init__(self):
```

```
        print("Rectangle constructor is called")
```

```
def displayRectangle(self):  
    print("Rectangle is displaying")
```

```
# Create objects of Rectangle class
```

```
rec = Rectangle()
```

```
# Call the methods
```

```
rec.displayShape()
```

```
rec.displayRectangle()
```

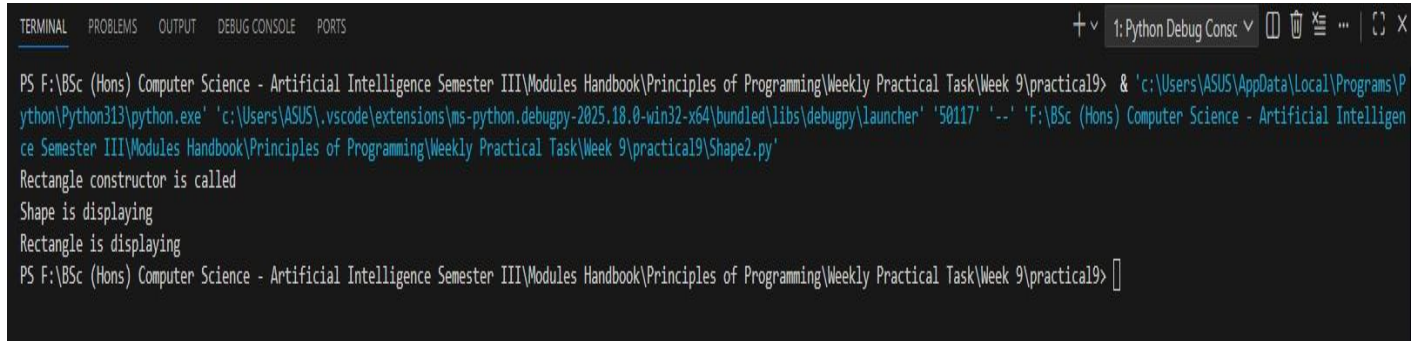
```
# Output:
```

```
# Rectangle constructor is called
```

```
# Shape is displaying
```

```
# Rectangle is displaying
```

## Output obtained in execution:



```

PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '50117' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape2.py'
Rectangle constructor is called
Shape is displaying
Rectangle is displaying
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> 

```

3.

Task 1.3. (Use of super) Modify the above program by calling Shape constructor in the Rectangle class using ‘super’ as below, save, compile, and run it.

Given code:

class Shape:

    #constructor for Shape class

    def \_\_init\_\_(self):

        print("Shape constructor is called")

    def displayShape(self):

        print("Shape is displaying")

```
class Rectangle(Shape):  
    # constructor for Rectangle class, Base class constructor is called using '  
    super'  
  
    def __init__(self):  
        super().__init__()  
        print("Rectangle constructor is called")  
  
    def displayRectangle(self):  
        print("Rectangle is displaying")  
  
rec = Rectangle()  
rec.displayShape()  
rec.displayRectangle()
```

If you can successfully run your program, you should see the following output:

Shape constructor is called

Rectangle constructor is called

Shape is displaying

Rectangle is displaying

Can you explain what's happening here and why? (Just write as a comment in the code).

**Answer:**

**Following code for input:**

```
class Shape:

    # Constructor for Shape class

    def __init__(self):

        print("Shape constructor is called")


    def displayShape(self):

        print("Shape is displaying")


class Rectangle(Shape):

    # constructor for Rectangle class, Base class constructor is called using 'super'

    def __init__(self):

        super().__init__()
```

```
print("Rectangle constructor is called")
```

```
def displayRectangle(self):
```

```
    print("Rectangle is displaying")
```

# Output:

# Shape constructor is called

# Rectangle constructor is called

# Shape is displaying

# Rectangle is displaying

### Output obtained in execution:



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '59041' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape3.py'
Shape constructor is called
Rectangle constructor is called
Shape is displaying
Rectangle is displaying
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> 
```

# Explanation of the output:

# First line of the output: "Shape constructor is called"

# Reason: Because inside the Rectangle.\_\_init(), we wrote super().\_\_init\_\_() which tell Python to first run the parent's constructor.

# Second line of the output: "Rectangle constructor is called"

# Reason: The rest of Rectangle.\_\_init\_\_() after super() finished

# Third line of the output: "Shape is displaying"

# Reason: It is called from rec.displayShape() where displayShape() exists in the parent class (Shape). Class Rectangle inherited it, so it can be called on Rectangle objects

# Fourth line of the output: "Rectangle is displaying"

# Reason : This line is called from rec.displayRectangle(). The method is present in the Rectangle class, not Shape.

4.

Task 1.4. (Accessing base class public attributes in the subclass) Modify the above program by creating a public attribute “colour” in the base class and access it in the subclass as below, save, compile, and run it.

Given code:

```
class Shape:

    # constructor for Shape class

    def __init__(self, colour='Black'):

        self.colour = colour

        print("Shape constructor is called")

    def displayShape(self):

        print("Shape is displaying")

    def getColour(self):

        return self.colour
```

```
class Rectangle(Shape):  
    # constructor for Rectangle class, Base class constructor is called using 'super'  
    def __init__(self):  
        super().__init__()   
        print("Rectangle constructor is called")  
  
    def displayRectangle(self):  
        print("Rectangle is displaying")  
  
    def getRectangleColour(self):  
        return self.colour  
  
rec = Rectangle()  
print(rec.getRectangleColour())
```

If you can successfully run your program, you should see the following output:

Shape constructor is called

Rectangle constructor is called

Black

Observation. Since 'colour' is a public member variable in the Shape class, it can be accessed anywhere, including in the subclass.

**Answer:**

**Following code for input:**

```
# Accessing base public attribute in subclass
```

```
# Practical 9 - Exercise 1 (Task 1.4)
```

```
# Nirvik K.C.
```

```
class Shape:
```

```
    # constructor for Shape class
```

```
    def __init__(self, colour='Black'):
```

```
        self.colour = colour
```

```
        print("Shape constructor is called")
```

```
    def displayShape(self):
```

```
        print("Shape is displaying")
```

```
    def getColour(self):
```

```
        return self.colour
```

```
class Rectangle(Shape):  
    # constructor for Rectangle class  
    # Parent class constructor is called using super()  
    def __init__(self):  
        super().__init__()  
        print("Rectangle constructor is called")  
  
    def displayRectangle(self):  
        print("Rectangle is displaying")  
  
    def getRectangleColour(self):  
        return self.colour # accessed the public attribute from parent  
  
# Create Rectangle object  
rec = Rectangle()  
  
# Print the colour which is accessed from the public attribute  
print(rec.getRectangleColour())
```

# Output:

# Shape constructor is called

# Rectangle constructor is called

# Black

### Output obtained in execution:



```

PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '58447' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape4.py'
Shape constructor is called
Rectangle constructor is called
Black
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>

```

5.

Task 1.5: Now, in your code change the access modifier of “colour” variable according to the following and observe the corresponding results/outputs

(a) Task 1.5: self.\_\_colour = colour # use a double underscore prefix

That is, make the variable as private and test your program as before if you can access it from the subclass.

Task 1.6 :

(b) `self._colour = colour` # use a single underscore prefix

That is, make the variable as protected and test your program as before if you can access it from the subclass.

Observation. Note that, a subclass does not inherit the private members of its base class. However, if the base class has public or protected methods for accessing its private fields, these can also be used by the subclass. Therefore, in this case you can use the following `getRectangleColour()` implementation to make it work.

```
def getRectangleColour(self):  
    return self.getColour()
```

(a) Task 1.5: `self.__colour = colour` # use a double underscore prefix

That is, make the variable as private and test your program as before if you can access it from the subclass.

**Answer:**

**Following code for input:**

# Testing private variable (double underscore) in inheritance

# Practical 9 - Exercise 1 - Task 1.5 (a)

# Nirvik K.C.

class Shape:

# Constructor - change the access modifier of “colour” variable

def \_\_init\_\_(self, colour='Black'):

# (a) Private attributes - use of double underscore

self.\_\_colour = colour

print("Shape constructor is called")

# (b) Protected version - use of single underscore

# self.\_colour = colour

# print("Shape constructor is called")

def displayShape(self):

print("Shape is displaying")

```
# Public getter method read the private variable
```

```
def getColour(self):
```

```
    return self.__colour
```

```
class Rectangle(Shape):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        print("Rectangle constructor is called")
```

```
    def displayRectangle(self):
```

```
        print("Rectangle is displaying")
```

```
# This method tries to access colour
```

```
def getRectangleColour(self):
```

```
    # Direct access to private variable using parent's public getter
```

```
    return self.getColour()
```

```
# Create Rectangle object
```

```
rec = Rectangle()
```

```
# Using inherited method
```

```
rec.displayShape()
```

```
# Using Rectangle's own method
```

```
rec.displayRectangle()
```

```
# Print the colour
```

```
print("Colour: ", rec.getRectangleColour())
```

```
# Direct access of the private attributes does not change the value
```

```
rec.__colour = "Red"
```

```
# Print the colour after change - Still Black
```

```
print("New Colour: ", rec.getRectangleColour())
```

# Output:

# Shape constructor is called

# Rectangle constructor is called

# Shape is displaying

# Rectangle is displaying

# Colour: Black

# New Colour: Black

### Output obtained in execution:



```
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '54146' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape6.py'
Shape constructor is called
Rectangle constructor is called
Shape is displaying
Rectangle is displaying
Colour: Black
New Colour: Black
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>
```

(b) 1.6: `self._colour = colour` # use a single underscore prefix

That is, make the variable as protected and test your program as before if you can access it from the subclass.

**Answer:**

**Following code for input:**

```
# Testing protected member variable with single underscore
```

```
# Practical 9 - Exercise 1 - Task 1.5 (b)
```

```
# Nirvik K.C.
```

```
class Shape:
```

```
    # Constructor - change the access modifier of “colour” variable
```

```
    def __init__(self, colour = 'Black'):
```

```
        # (b) Protected attributes - use of single underscore
```

```
        self._colour = colour
```

```
        print("Shape constructor is called")
```

```
# (a) Private attributes - use of double underscore

# self.__colour = colour

# print("Shape constructor is called")


def displayShape(self):

    print("Shape is displaying")


# Public getter used to read the protected variable

def getColour(self):

    return self.__colour


class Rectangle(Shape):

    def __init__(self):

        super().__init__()

        print("Rectangle constructor is called")


def displayRectangle(self):

    print("Rectangle is displaying")
```

```
# Method which accessed the protected variable 'colour' directly

def getRectangleColour(self):

    return self._colour


# Create Rectangle object

rec = Rectangle()


# Using inherited method

rec.displayShape()


# Using Rectangle's own method

rec.displayRectangle()


# Print the colour

print("Colour: ", rec.getRectangleColour())


# Direct access of the private attributes does not change the value

rec._colour = "Red"
```

# Print the colour after change - Still Black

```
print("New Colour: ", rec.getRectangleColour())
```

# Output:

# Shape constructor is called

# Rectangle constructor is called

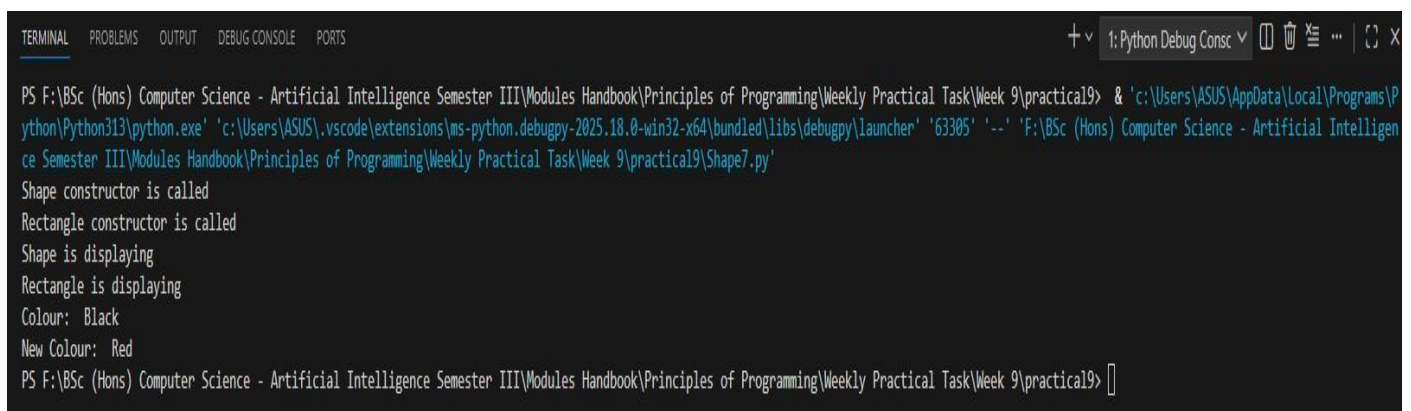
# Shape is displaying

# Rectangle is displaying

# Colour: Black

# New Colour: Red

### Output obtained in execution:



```
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '63305' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape7.py'
Shape constructor is called
Rectangle constructor is called
Shape is displaying
Rectangle is displaying
Colour: Black
New Colour: Red
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>
```

7.

Task 1.7: (Use of Polymorphism: overriding method) Modify the above program according to the following:

In the Shape class, change the displayShape(self) method name to display(self) In the Rectangle class, change the displayShape(self) method name to display(self) Then call the display() method using a rectangle object as below.

Given code:

```
class Shape:
```

```
    #constructor for Shape class
```

```
    def __init__(self, colour='Black'):
```

```
        self.colour = colour
```

```
        print("Shape constructor is called")
```

```
    def display(self):
```

```
        print("Shape is displaying")
```

```
    def getColour(self):
```

```
        return self.colour
```

```
class Rectangle(Shape):  
    # constructor for Rectangle class, Base class constructor is called using '  
    super'  
  
    def __init__(self):  
        super().__init__()  
        print("Rectangle constructor is called")  
  
    def display(self):  
        print("Rectangle is displaying")  
  
    def getRectangleColour(self):  
        return self.getColour()  
  
rec = Rectangle()  
print(rec.getRectangleColour())  
rec.display()
```

If you can successfully run your program, you should see the following output:

Shape constructor is called

Rectangle constructor is called

Black

Rectangle is displaying

**Answer:**

**Following code for input:**

```
# Use of Polymorphism: overriding method
```

```
# Practical 9 - Exercise 1 - Task 1.7
```

```
# Nirvik K.C.
```

```
class Shape:
```

```
    # Constructor for Shape Class
```

```
    def __init__(self, colour='Black'):
```

```
        self.colour = colour
```

```
        print("Shape constructor is called")
```

```
def display(self):  
    print("Shape is displaying")
```

```
def getColour(self):  
    return self.colour
```

```
class Rectangle(Shape):  
    # Constructor for Rectangle class  
    def __init__(self):  
        super().__init__()  
        print("Rectangle constructor is called")  
  
    def display(self):  
        print("Rectangle is displaying")  
  
    def getRectangleColour(self):  
        return self.getColour()
```

```
# Create Rectangle object
```

```
rec = Rectangle()
```

```
# Print the colour using getter
```

```
print(rec.getRectangleColour())
```

```
# Call display() on Rectangle object
```

```
rec.display()
```

```
# Output:
```

```
# Shape constructor is called
```

```
# Rectangle constructor is called
```

```
# Black
```

```
# Rectangle is displaying
```

## Output obtained in execution:

```

PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '51673' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape8.py'
Shape constructor is called
Rectangle constructor is called
Black
Rectangle is displaying
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>

```

Observation: Note that both versions of the `display()` method are members of the `Rectangle` class. However, in the output you see `Rectangle is displaying` (and not the `Shape is displaying`).

Can you explain, why? (Just write as a comment in the code).

Answer:

# Explanation of output order and why "Rectangle is displaying" is shown (not "Shape is displaying")

# Reason: This is an example of method overriding in polymorphism.

# The `Rectangle` class inherits from `Shape` and overrides the `display()` method.

# When `rec.display()` is called on a `Rectangle` object, Python looks for the method in the `Rectangle` class first.

# Since Rectangle has its own display() method, it executes that instead of the one in Shape.

# This is why "Rectangle is displaying" is printed instead of "Shape is displaying."

8.

Task 1.8: (Use of Polymorphism: overriding method) Now create another Square subclass of the Shape class with an empty body according to the following. Create an instance of the Square class and call the display() method using that object. Run your code and explain the output to your tutors.

Given code:

```
class Shape:
```

```
    #constructor for Shape class
```

```
    def __init__(self, colour='Black'):
```

```
        self.colour = colour
```

```
        print("Shape constructor is called")
```

```
    def display(self):
```

```
        print("Shape is displaying")
```

```
def getColour(self):
```

```
    return self.colour
```

```
class Rectangle(Shape):
```

```
    #constructor for Rectangle class, Base class constructor is called using '
```

```
super'
```

```
def __init__(self):
```

```
    super().__init__()
```

```
    print("Rectangle constructor is called")
```

```
def display(self):
```

```
    print("Rectangle is displaying")
```

```
def getRectangleColour(self):
```

```
    return self.getColour()
```

```
class Square(Shape):
```

```
    pass
```

```
rec = Rectangle()

print(rec.getRectangleColour())

rec.display()

sqr = Square()

sqr.display()
```

**Answer:**

**Following code for input:**

```
# Polymorphism & overriding with empty subclass

# Practical 9 - Exercise 1 - Task 1.8

# Nirvik K.C.
```

```
class Shape:

    # Constructor for Shape class

    def __init__(self, colour='Black'):

        self.colour = colour

        print("Shape constructor is called")
```

```
def display(self):  
    print("Shape is displaying")
```

```
def getColour(self):  
    return self.colour
```

```
class Rectangle(Shape):  
    # Constructor for Rectangle class  
    def __init__(self):  
        super().__init__()  
        print("Rectangle constructor is called")  
  
    # Overrides the display() method from Shape  
    def display(self):  
        print("Rectangle is displaying")  
  
    def getRectangleColour(self):  
        return self.getColour()
```

```
class Square(Shape):  
    # Empty body with no constructor or method defined  
  
    # Inherits everything from Shape class  
  
    pass  
  
# Create a Rectangle object  
rec = Rectangle()  
  
# Call display() on Rectangle object - uses Rectangle's overridden method  
rec.display()  
  
# Create a Square object  
sqr = Square()  
  
# Call display() on Square object - inherits method from Shape  
sqr.display()
```

# Output:

# Shape constructor is called

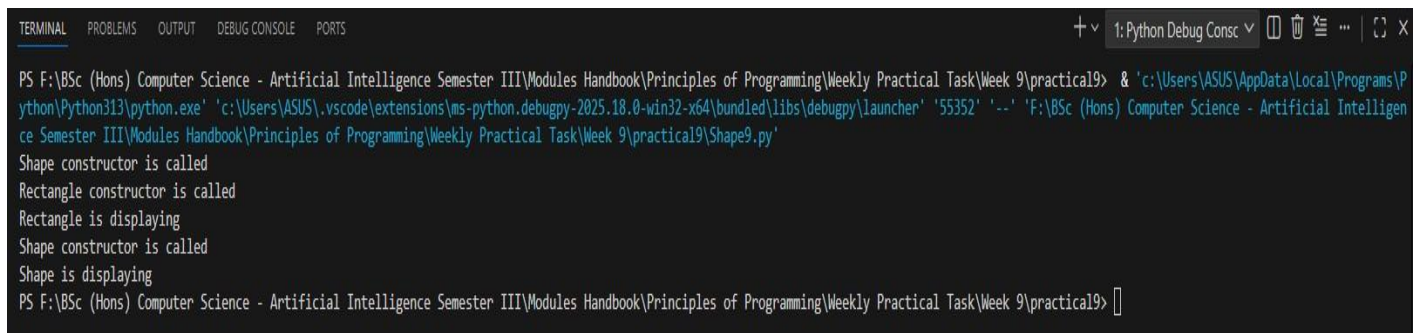
# Rectangle constructor is called

# Rectangle is displaying

# Shape constructor is called

# Shape is displaying

### Output obtained in execution:



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
+ v  1: Python Debug Consc  [icon] [icon] [icon] [icon] [icon] [icon] [icon] [icon] [icon] [icon]
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55352' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Shape9.py'
Shape constructor is called
Rectangle constructor is called
Rectangle is displaying
Shape constructor is called
Shape is displaying
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> [ ]
```

9.

Task 1.9: (Use of Polymorphism: overriding method) Now using your Square object call the `getRectangleColour()` method of the rectangle class as below. Run your code and explain the output to your tutors.

Given code:

```
rec = Rectangle()

print(rec.getRectangleColour())

rec.display()

sqr = Square()

sqr.display()

sqr.getRectangleColour()
```

**Answer:**

**Following code for input:**

```
# Polymorphism & calling subclass method on unrelated object

# Practical 9 - Exercise 1 - Task 1.9

# Nirvik K.C.
```

```
class Shape:

    # Constructor for Shape class

    def __init__(self, colour='Black'):

        self.colour = colour

        print("Shape constructor is called")


    def display(self):

        print("Shape is displaying")


    def getColour(self):

        return self.colour


class Rectangle(Shape):

    # Constructor for Rectangle class

    def __init__(self):

        super().__init__()

        print("Rectangle constructor is called")
```

```
# Overrides the display() method from Shape

def display(self):

    print("Rectangle is displaying")


def getRectangleColour(self):

    return self.getColour()


class Square(Shape):

    # Empty body with no constructor or method defined

    # Inherits everything from Shape class

    pass


# Create a Rectangle object

rec = Rectangle()


# Call the getRectangleColour() method of the rectangle class

print(rec.getRectangleColour())
```

# Call display() on Rectangle object - uses Rectangle's overridden method

rec.display()

# Create a Square object

sqr = Square()

# Call display() on Square object - inherits method from Shape

sqr.display()

# Call method - getRectangleColour() on Square object

sqr.getRectangleColour()

# Output(until the error):

# Shape constructor is called

# Rectangle constructor is called

# Black

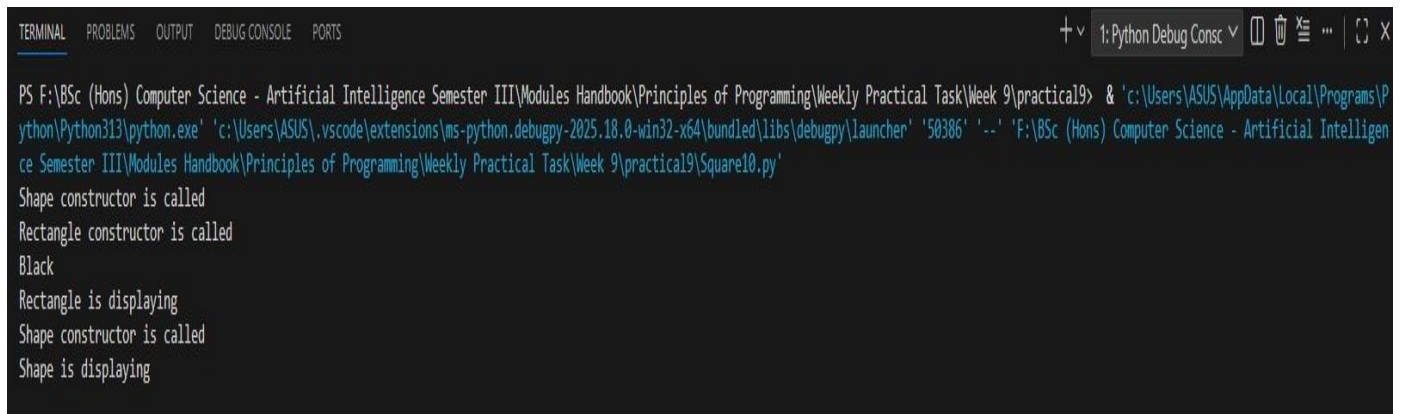
# Rectangle is displaying

# Shape constructor is called

# Shape is displaying

# AttributeError: 'Square' object has no attribute 'getRectangleColour'

### Output obtained in execution (until the error):

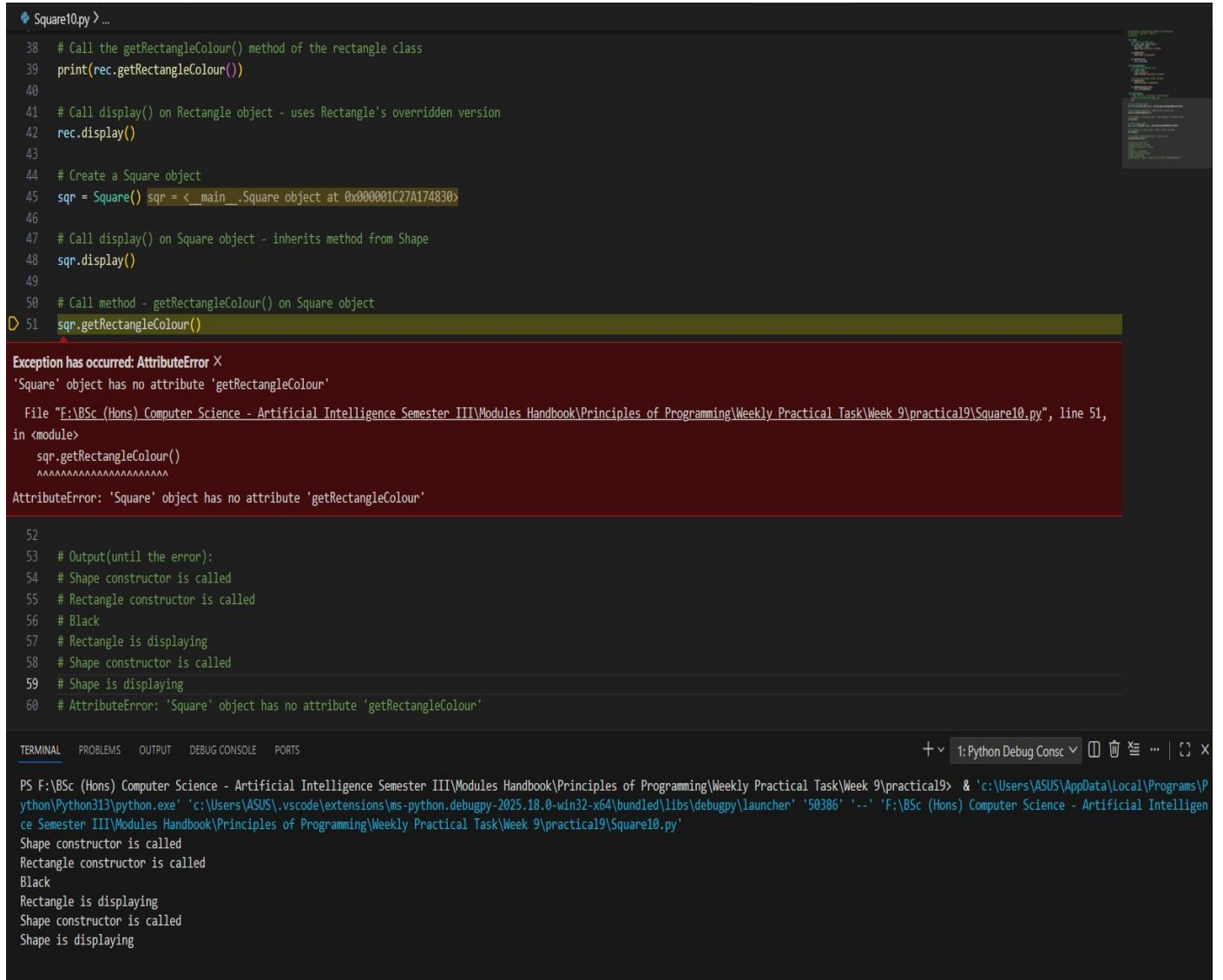


```
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '50386' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Square10.py'
Shape constructor is called
Rectangle constructor is called
Black
Rectangle is displaying
Shape constructor is called
Shape is displaying
```

The Error Message:

AttributeError: 'Square' object has no attribute 'getRectangleColour'

## The Message Shown:



```

Square10.py > ...
38 # Call the getRectangleColour() method of the rectangle class
39 print(rec.getRectangleColour())
40
41 # Call display() on Rectangle object - uses Rectangle's overridden version
42 rec.display()
43
44 # Create a Square object
45 sqr = Square()  <__main__.Square object at 0x000001C27A174830>
46
47 # Call display() on Square object - inherits method from Shape
48 sqr.display()
49
50 # Call method - getRectangleColour() on Square object
51 sqr.getRectangleColour()

Exception has occurred: AttributeError
'Square' object has no attribute 'getRectangleColour'

File "F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Square10.py", line 51,
in <module>
    sqr.getRectangleColour()
    ~~~~~^
AttributeError: 'Square' object has no attribute 'getRectangleColour'

52
53 # Output(until the error):
54 # Shape constructor is called
55 # Rectangle constructor is called
56 # Black
57 # Rectangle is displaying
58 # Shape constructor is called
59 # Shape is displaying
60 # AttributeError: 'Square' object has no attribute 'getRectangleColour'

```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS

PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '50386' '-' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Square10.py'

Shape constructor is called  
Rectangle constructor is called  
Black  
Rectangle is displaying  
Shape constructor is called  
Shape is displaying

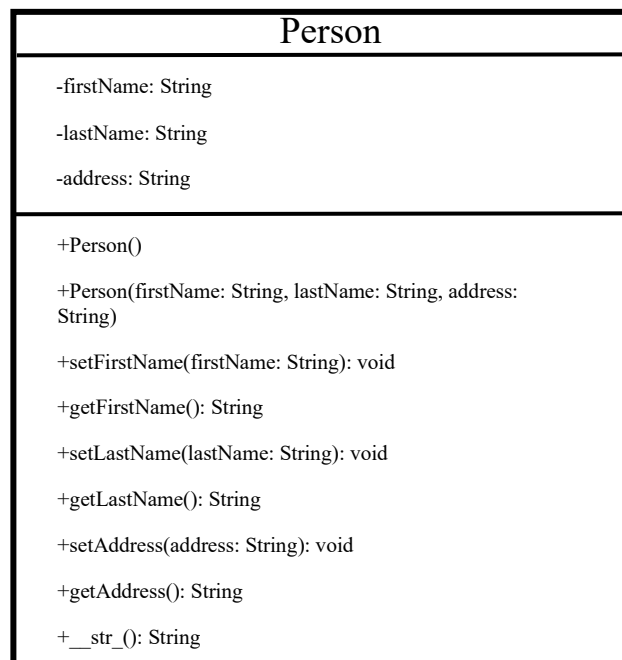
Explain the output and error:

When `rec = Rectangle()` is executed, the `Rectangle` constructor runs first and calls `super().__init__()`, which triggers the `Shape` constructor. This prints out “Shape constructor is called”. After the parent finishes, the `Rectangle` constructor continues and prints out “Rectangle constructor is called.” Then `rec.getRectangleColour()` return ‘Black’ (set in `Shape`). Finally, `rec.display()` uses `Rectangle`’s overridden method that prints out “Rectangle is displaying.” The `Square()` method has no `display`, so it uses inherited `display()` from `Shape` which prints out “Shape is displaying”. The `sqr.getRectangleColour()` method exist only in `Rectangle` class and `Square` does not inherit it from `Rectangle` (only from `Shape`). So, python cannot find the `getRectangleColour()` which results in `AttributeError`. Therefore, methods defined in one subclass (`Rectangle`) are not automatically available in another subclass (`Square`).

## Exercise 2: Design and Implementation

Your starting point will be to consider a simple example based on people. Suppose you are developing a software system for UWE. They have already designed, written, coded and tested a class called Person. An object in this class holds personal information about somebody using his name and address. A Person also has methods that can construct a new person from given data, print the information in a standard format, set the name and address and so on. Now UWE also wants a Lecturer class, a Student class, and a GraduateStudent class that are all different kinds of People!

A UML class diagram of the Person class and its corresponding Python code implementation are shown below. Note that all the member variables are private and methods are public.



Given code:

```
class Person:
```

```
    # constructor
```

```
    def __init__(self, firstName='Mr', lastName='X', address='Bristol'):
```

```
        self.__firstName = firstName #private variable
```

```
        self.__lastName = lastName  #private variable
```

```
        self.__address = address    #private variable
```

```
    def setFirstName(self, firstName):
```

```
        self.__firstName = firstName
```

```
    def setLastName(self, lastName):
```

```
        self.__lastName = lastName
```

```
    def setAddress(self,address):
```

```
        self.__address = address
```

```
def getFirstName(self):  
    return self.__firstName  
  
def getLastName(self):  
    return self.__lastName  
  
def getAddress(self):  
    return self.__address  
  
def __str__(self):  
    return "Person's name is " + self.getFirstName() + " " + self.getLastName()  
    e() + "\nAddress is " + self.getAddress()  
  
# creating a Person object with default values  
print("Creating a Person object with default values")  
  
p = Person()  
  
# printing the information using __str__ method  
print("Printing the information using __str__ method")  
  
print(p)
```

```
# setting name and address

print("Now setting the names and address")

p.setFirstName("Abdur")

p.setLastName("Rakib")

p.setAddress("UWE Frenchay Campus 4QXX")

#printing the information using __str__ method

print("Printing the information using __str__ method")

print(p)

#Creating another object by passing the values explicitly

print("Creating another Person object by passing the values explicitly")

p1 = Person("Jun", "Hong", "UWE Frenchay Campus 3QXX")

#printing the information using __str__ method

print("Printing the information using __str__ method")

print(p1)
```

10. Task 2.1: Study the Person class diagram and the corresponding Python code.  
Write, save, compile, and run the above code.

/\* file name should be Person.py\*/

**Answer:**

**Following code for input:**

```
# Person class with private attributes and __str__

# Practical 9 - Exercise 2 - Task 2.1

# Nirvik K.C.

class Person:

    # Constructor with default values

    def __init__(self, firstName='Mr', lastName='X', address='Bristol'):

        self.__firstName = firstName # private variable

        self.__lastName = lastName # private variable

        self.__address = address # private variable


    # Setters

    def setFirstName(self, firstName):

        self.__firstName = firstName


    def setLastName(self, lastName):

        self.__lastName = lastName
```

```
def setAddress(self, address):  
    self.__address = address  
  
# Getters  
  
def getFirstName(self):  
    return self.__firstName  
  
def getLastName(self):  
    return self.__lastName  
  
def getAddress(self):  
    return self.__address  
  
# Use of __str__ method  
  
def __str__(self):  
    return "Person's name is " + self.getFirstName() + " " + self.getLastName()  
    + "\nAddress is " + self.getAddress()
```

```
# Test program

# Creating a Person object with default values

print("Creating a Person object with default values")

p = Person()

# Printing the information using __str__ method

print("Printing the information using __str__ method")

print(p)

# Setting name and address

print("Now setting the names and address")

p.setFirstName("Abdur")

p.setLastName("Rakib")

p.setAddress("UWE Frenchay Campus 4QXX")
```

```
# Printing the information using __str__ method
print("Printing the information using __str__ method")

print(p)

# Creating another object by passing values explicitly
print("Creating another Person object by passing the values explicitly")

p1 = Person("Jun", "Hong", "UWE Frenchay Campus 3QXX")

# Printing the information using __str__ method
print("Printing the information using __str__ method")

print(p1)

# Output:

# Creating a Person object with default values

# Printing the information using __str__ method

# Person's name is Mr X

# Address is Bristol

# Now setting the names and address
```

```
# Printing the information using __str__ method

# Person's name is Abdur Rakib

# Address is UWE Frenchay Campus 4QXX

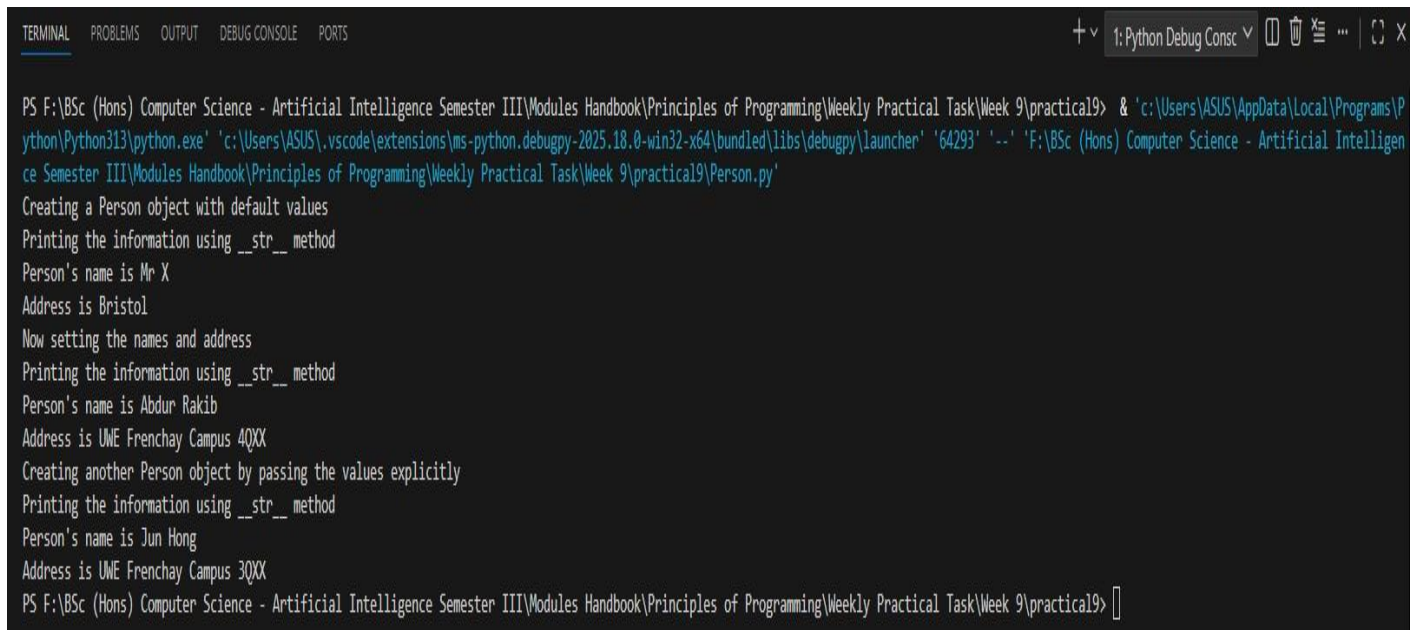
# Creating another Person object by passing the values explicitly

# Printing the information using __str__ method

# Person's name is Jun Hong

# Address is UWE Frenchay Campus 3QXX
```

### Output obtained in execution:



```
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64293' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Person.py'
Creating a Person object with default values
Printing the information using __str__ method
Person's name is Mr X
Address is Bristol
Now setting the names and address
Printing the information using __str__ method
Person's name is Abdur Rakib
Address is UWE Frenchay Campus 4QXX
Creating another Person object by passing the values explicitly
Printing the information using __str__ method
Person's name is Jun Hong
Address is UWE Frenchay Campus 3QXX
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>
```

11.

Task 2.2: Design three classes Lecturer, Student, and GraduateStudent using pen and paper (or you can use MS Word/PowerPoint editor). The classes Lecturer and Student extend/inherit Person, and GraduateStudent extends/inherits Student. Draw the complete UML class diagram considering all the four classes. Your three newly designed classes contain the following:

Lecturer class:

- One private integer data field representing lecturer ID
- Appropriate constructor that creates a lecturer with the specified name, address, and the lecturer ID (note that you need to access name and address from the base class)
- All the appropriate 'set' and 'get' methods
- A method named `__str__()` that returns a string description for the lecturer. It will use the base class `__str__()`

Student class:

- One private integer data field representing student ID

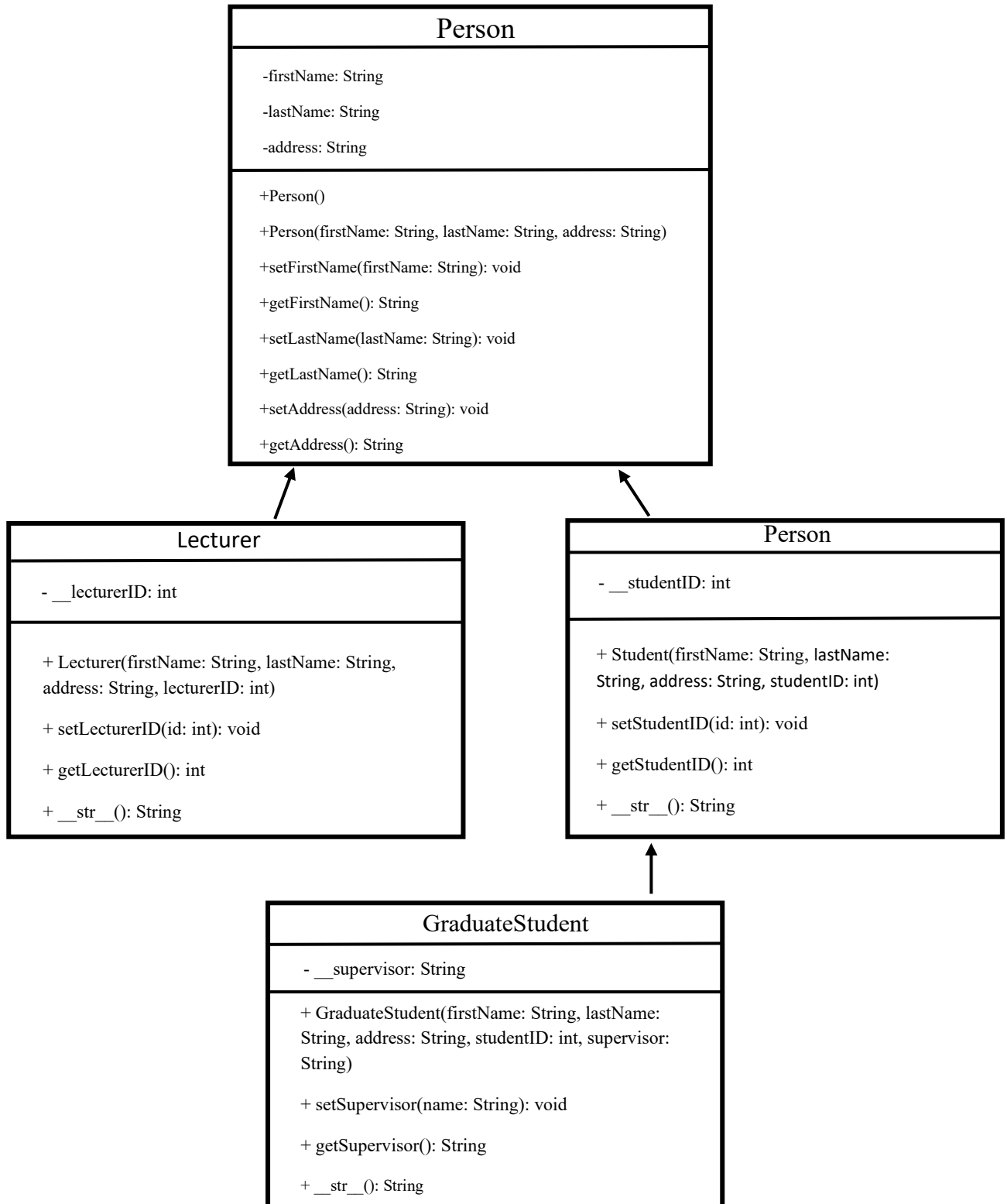
- Appropriate constructor that creates a student with the specified name, address, and the student ID (note that you need to access name and address from the base class)
- All the appropriate ‘set’ and ‘get’ methods
- A method named `__str__()` that returns a string description for the lecturer. It will use its base class `__str__()`

GraduateStudent class:

- One private String data field representing supervisor’s name (usually, a graduate student works under an assigned supervisor)
- Appropriate constructor that creates a graduate student with the specified name, address, the student ID, and the supervisor’s name (note that you need to access name, address, and student ID from the base class)
- All the appropriate ‘set’ and ‘get’ methods
- A method named `__str__()` that returns a string description for the lecturer. It will use its base class `__str__()`

**Answer:**

**UML Diagram for the all four classes:**



**Following code for input:**

```
# Inheritance with Person, Lecturer, Student, GraduateStudent
```

```
# Practical 9 - Exercise 2 - Task 2.2
```

```
# Nirvik K.C.
```

```
class Person:
```

```
    def __init__(self, firstName="Mr", lastName="X", address="Bristol"):
```

```
        self.__firstName = firstName
```

```
        self.__lastName = lastName
```

```
        self.__address = address
```

```
    # Setters
```

```
    def setFirstName(self, firstName):
```

```
        self.__firstName = firstName
```

```
    def setLastName(self, lastName):
```

```
        self.__lastName = lastName
```

```
def setAddress(self, address):  
    self.__address = address  
  
# Getters  
  
def getFirstName(self):  
    return self.__firstName  
  
def getLastName(self):  
    return self.__lastName  
  
def getAddress(self):  
    return self.__address  
  
# __str__ method  
  
def __str__(self):  
    return f'Person's name is {self.getFirstName()}  
{self.getLastName()}\nAddress is {self.getAddress()}'
```

```
class Lecturer(Person):  
  
    def __init__(self, firstName, lastName, address, lecturerID):  
        super().__init__(firstName, lastName, address)  
        self.__lecturerID = lecturerID  
  
    # Getters and setters for lecturerID  
  
    def getLecturerID(self):  
        return self.__lecturerID  
  
    def setLecturerID(self, lecturerID):  
        self.__lecturerID = lecturerID  
  
    # Override __str__  
  
    def __str__(self):  
        return super().__str__() + f"\nLecturer ID: {self.getLecturerID()}"
```

```
class Student(Person):

    def __init__(self, firstName, lastName, address, studentID):

        super().__init__(firstName, lastName, address)

        self.__studentID = studentID


# Getters and setters for studentID

def getStudentID(self):

    return self.__studentID


def setStudentID(self, studentID):

    self.__studentID = studentID


# Override __str__

def __str__(self):

    return super().__str__() + f"\nStudent ID: {self.getStudentID()}"
```

```
class GraduateStudent(Student):

    def __init__(self, firstName, lastName, address, studentID, supervisor):

        super().__init__(firstName, lastName, address, studentID)

        self.__supervisor = supervisor


# Getters and setters for supervisor

def getSupervisor(self):

    return self.__supervisor


def setSupervisor(self, supervisor):

    self.__supervisor = supervisor


# Override __str__

def __str__(self):

    return super().__str__() + f"\nSupervisor: {self.getSupervisor()}"
```

```
# Lecturer
```

```
lec = Lecturer("Dr. Wilson", "Smith", "London", 101)
```

```
print(lec)
```

```
print()
```

```
# Student
```

```
stu = Student("Harry", "Conway", "Bristol", 239201)
```

```
print(stu)
```

```
print()
```

```
# GraduateStudent
```

```
grad = GraduateStudent("Joe", "Clark", "Manchester", 2023002, "Prof. David")
```

```
print(grad)
```

## Output obtained in execution:

```

PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '64480' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Person2.py'
Person's name is Dr. Wilson Smith
Address is London
Lecturer ID: 101

Person's name is Harry Conway
Address is Bristol
Student ID: 239201

Person's name is Joe Clark
Address is Manchester
Student ID: 2023002
Supervisor: Prof. David
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9>

```

12.

Task 2.3: Implement your classes, create appropriate objects and print the information. For example, you should create objects and print the information like this:

```
lecturer = Lecturer ("Chris", " Simons", "UWE Frenchay Campus 4QXX",2345);
print(lecturer);
```

```
student = Student ("Peter", "Miller","UWE Frenchay Campus",5678);
print(student);
```

```
gradstudent = GraduateStudent ("Dan", "Fielding", "UWE Frenchay  
Campus",6789,"Jim Smith");  
  
print(gradstudent);
```

Output:

Lecturer's name is Chris Simons

Address is UWE Frenchay Campus 4QXX

Lecturer ID: 2345

Students's name is Peter Miller

Address is UWE Frenchay Campus

Student ID: 5678

Students's name is Dan Fielding

Address is UWE Frenchay Campus

Student ID: 6789

Supervisor's name: Jim Smith

**Answer:**

**Following code for input:**

# Implement your classes, create appropriate objects and print the information

# Practical 9 - Exercise 2 - Task 2.3

# Nirvik K.C.

class Person:

def \_\_init\_\_(self, firstName="Mr", lastName="X", address="Bristol"):

self.\_\_firstName = firstName

self.\_\_lastName = lastName

self.\_\_address = address

# Getters

def getFirstName(self):

return self.\_\_firstName

def getLastName(self):

return self.\_\_lastName

def getAddress(self):

return self.\_\_address

```

# __str__ method

def __str__(self):

    return f'{self.getFirstName()}'s name is {self.getFirstName()}
{self.getLastName()}\n" f'Address is {self.getAddress()}'"

class Lecturer(Person):

    def __init__(self, firstName, lastName, address, lecturerID):

        super().__init__(firstName, lastName, address)

        self.__lecturerID = lecturerID

# Override __str__

def __str__(self):

    return "Lecturer's name is " + self.getFirstName() + " " + self.getLastName()
+ "\n" "Address is " + self.getAddress() + "\n" "Lecturer ID: " +
str(self.__lecturerID)

```

```

class Student(Person):

    def __init__(self, firstName, lastName, address, studentID):

        super().__init__(firstName, lastName, address)

        self.__studentID = studentID


    # Override __str__

    def __str__(self):

        return "Student's name is " + self.getFirstName() + " " + self.getLastName() +
"\n" "Address is " + self.getAddress() + "\n" "Student ID: " + str(self.__studentID)


class GraduateStudent(Student):

    def __init__(self, firstName, lastName, address, studentID, supervisor):

        super().__init__(firstName, lastName, address, studentID)

        self.__supervisor = supervisor


    # Override __str__

    def __str__(self):

        return "Student's name is " + self.getFirstName() + " " + self.getLastName() +
"\n" "Address is " + self.getAddress() + "\n" + "Student ID: " +
str(self.__Student__studentID) + "\n" "Supervisor's name: " + self.__supervisor

```

```
# Lecturer
```

```
lecturer = Lecturer("Chris", "Simons", "UWE Frenchay Campus 4QXX", 2345)
```

```
print(lecturer)
```

```
# Student
```

```
student = Student("Peter", "Miller", "UWE Frenchay Campus", 5678)
```

```
print(student)
```

```
# GraduateStudent
```

```
gradstudent = GraduateStudent("Dan", "Fielding", "UWE Frenchay Campus",  
6789, "Jim Smith")
```

```
print(gradstudent)
```

```
# Output:
```

```
# Lecturer's name is Chris Simons
```

```
# Address is UWE Frenchay Campus 4QXX
```

```
# Lecturer ID: 2345
```

```
# Student's name is Peter Miller
```

# Address is UWE Frenchay Campus

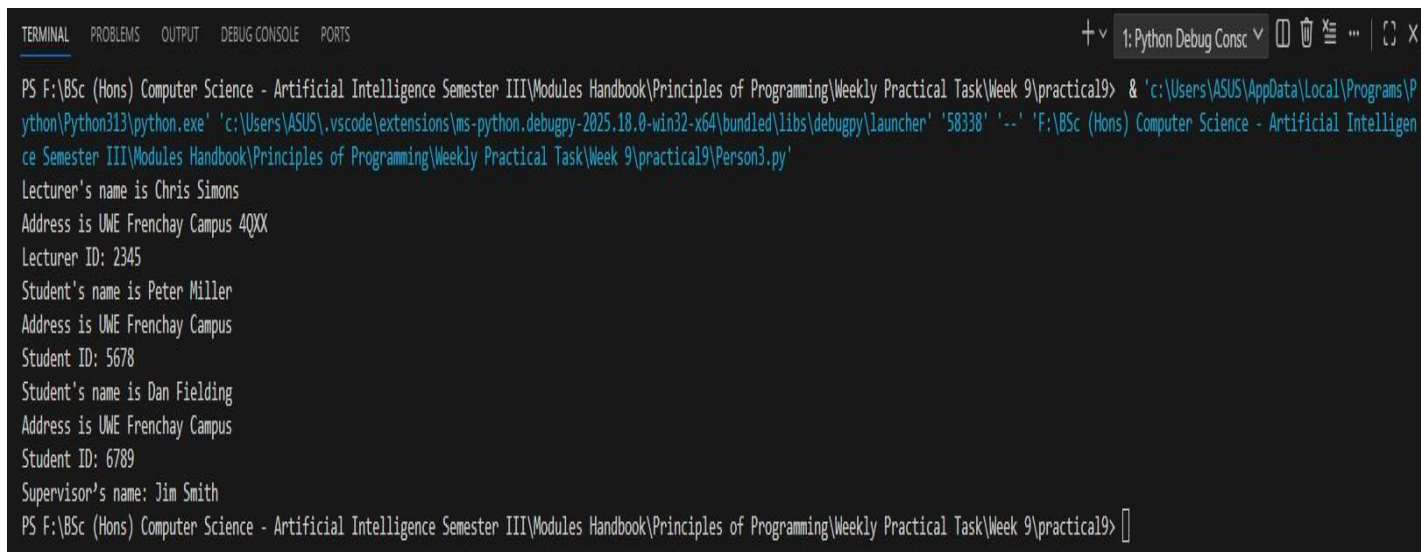
# Student ID: 5678

# Student's name is Dan Fielding

# Address is UWE Frenchay Campus

# Student ID: 6789

### Output obtained in execution:



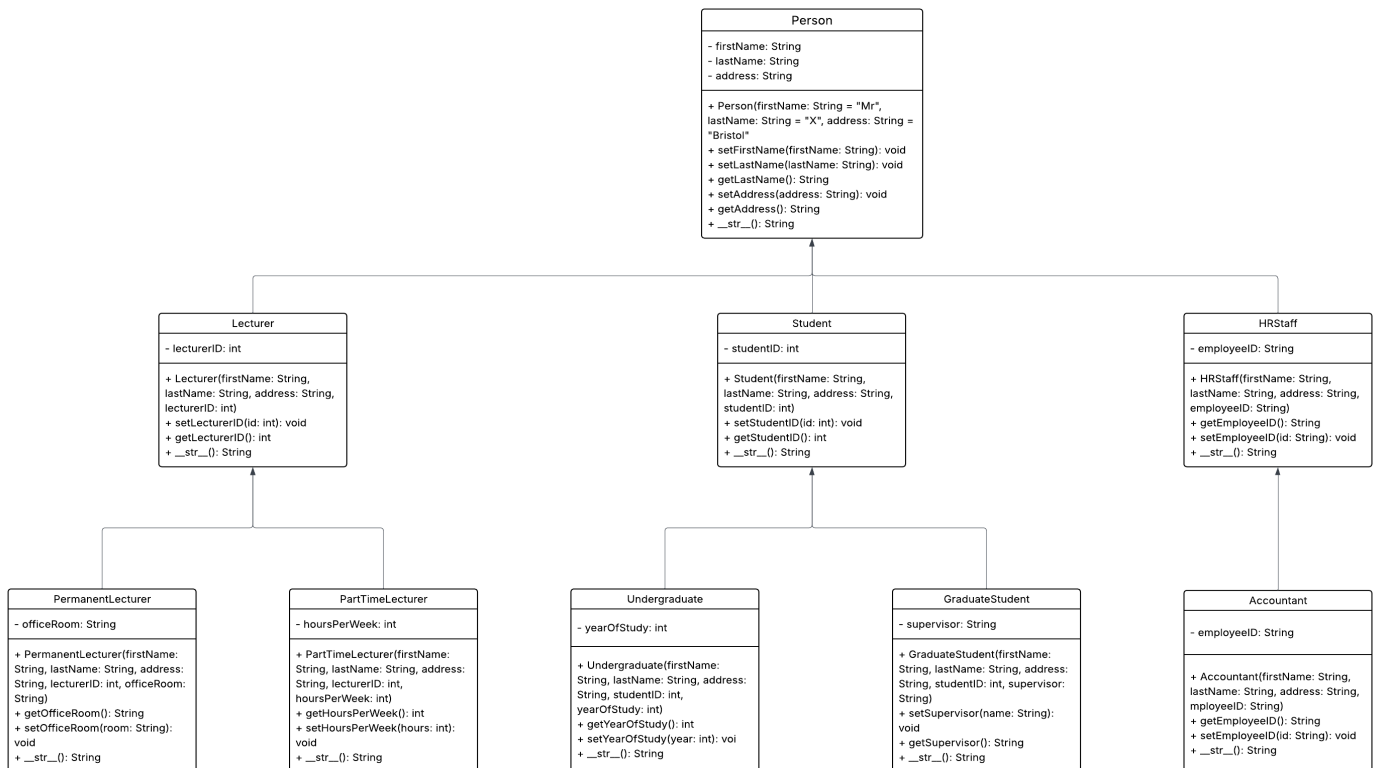
```
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical19> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58338' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical19\Person3.py'
Lecturer's name is Chris Simons
Address is UWE Frenchay Campus 4QXX
Lecturer ID: 2345
Student's name is Peter Miller
Address is UWE Frenchay Campus
Student ID: 5678
Student's name is Dan Fielding
Address is UWE Frenchay Campus
Student ID: 6789
Supervisor's name: Jim Smith
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical19> 
```

13.

Task 2.4. If you want to practice more, extend your design and code by adding new classes. For example, UnderGraduate (sub-class of Student), PermanentLecturer (sub-class of Lecturer), PartTimeLecturer (sub-class of Lecturer), HRStaff (sub-class of Person), and Accountant (sub-class of HRStaff). Draw the UML diagram for the classes and then implement them. Design them appropriately considering their appropriate member variables and methods.

**Answer:**

UML Diagram for all classes:



**Answer:**

**Following code for input:**

```
# Implementation of complete hierarchy for Person class

# Adding new classes.

# For example, UnderGraduate (sub-class of Student), PermanentLecturer (sub-
class of Lecturer),

# PartTimeLecturer (sub-class of Lecturer), HRStaff (sub-class of Person), and
Accountant (sub-class of HRStaff)
```

```
class Person:
```

```
    def __init__(self, firstName="Mr", lastName="X", address="Bristol"):
```

```
        self.__firstName = firstName
```

```
        self.__lastName = lastName
```

```
        self.__address = address
```

```
    # Getters
```

```
    def getFirstName(self):
```

```
        return self.__firstName
```

```
def getLastName(self):  
    return self.__lastName
```

```
def getAddress(self):  
    return self.__address
```

```
# __str__ method
```

```
def __str__(self):  
    return (f'Person's name is {self.getFirstName()} {self.getLastName()}\n'  
f'Address is {self.getAddress()}")
```

```
class Lecturer(Person):
```

```
    def __init__(self, firstName, lastName, address, lecturerID):  
        super().__init__(firstName, lastName, address)  
        self.__lecturerID = lecturerID
```

```

# Getters and setters

def getLecturerID(self):

    return self.__lecturerID


def setLecturerID(self, lecturerID):

    self.__lecturerID = lecturerID


def __str__(self):

    return "Lecturer's name is " + self.getFirstName() + " " + self.getLastName()
+ "\n" "Address is " + self.getAddress() + "\n" "Lecturer ID: " +
str(self.__lecturerID)


class PermanentLecturer(Lecturer):

    def __init__(self, firstName, lastName, address, lecturerID, officeRoom):

        super().__init__(firstName, lastName, address, lecturerID)

        self.__officeRoom = officeRoom

# Getters and setters

def getOfficeRoom(self):

    return self.__officeRoom

```

```
def setOfficeRoom(self, officeRoom):
```

```
    self.__office = officeRoom
```

```
def __str__(self):
```

```
    return "Lecturer's name is " + self.getFirstName() + " " + self.getLastName()  
+ "\n" "Address is " + self.getAddress() + "\n" "Lecturer ID: " +  
str(self.getLecturerID()) + "\n" "Office Room: " + self.__officeRoom
```

```
class PartTimeLecturer(Lecturer):
```

```
def __init__(self, firstName, lastName, address, lecturerID, hoursPerWeek):
```

```
    super().__init__(firstName, lastName, address, lecturerID)
```

```
    self.__hoursPerWeek = hoursPerWeek
```

```
def getHoursPerWeek(self):
```

```
    return self.__hoursPerWeek
```

```
def setHoursPerWeek(self, hours):
```

```
    self.__hoursPerWeek = hours
```

```
def __str__(self):  
    return "Lecturer's name is " + self.getFirstName() + " " + self.getLastName()  
+ "\n" "Address is " + self.getAddress() + "\n" "Lecturer ID: " +  
str(self.getLecturerID()) + "\n" "Hours per week: " + str(self.__hoursPerWeek)
```

```
class Student(Person):
```

```
    def __init__(self, firstName, lastName, address, studentID):  
        super().__init__(firstName, lastName, address)  
        self.__studentID = studentID
```

```
# Getters and setters
```

```
def getStudentID(self):  
    return self.__studentID
```

```
def setStudentID(self, studentID):  
    self.__studentID = studentID
```

```
# Override __str__

def __str__(self):

    return "Student's name is " + self.getFirstName() + " " + self.getLastName() +
"\n" "Address is " + self.getAddress() + "\n" "Student ID: " + str(self.__studentID)

class Undergraduate(Student):

    def __init__(self, firstName, lastName, address, studentID, yearOfStudy):

        super().__init__(firstName, lastName, address, studentID)

        self.__yearOfStudy = yearOfStudy

# Getters and setters

def getYearOfStudy(self):

    return self.__yearOfStudy

def setYearOfStudy(self, year):

    self.__yearOfStudy = year
```

```

def __str__(self):

    return "Student's name is " + self.getFirstName() + " " + self.getLastName() +
"\n" "Address is " + self.getAddress() + "\n" "Student ID: " +
str(self.getStudentID()) + "\n" "Year of Study: " + str(self.__yearOfStudy)

```

```

class GraduateStudent(Student):

```

```

    def __init__(self, firstName, lastName, address, studentID, supervisor):

        super().__init__(firstName, lastName, address, studentID)

        self.__supervisor = supervisor

    # Getters and setters

    def getSupervisor(self):

        return self.__supervisor

    def setSupervisor(self, supervisor):

        self.__supervisor = supervisor


    def __str__(self):

        return "Student's name is " + self.getFirstName() + " " + self.getLastName() +
"\n" "Address is " + self.getAddress() + "\n" "Student ID: " +
str(self.getStudentID()) + "\n" "Supervisor's name: " + self.__supervisor

```

```
class HRStaff(Person):

    def __init__(self, firstName, lastName, address, employeeID):

        super().__init__(firstName, lastName, address)

        self.__employeeID = employeeID


# Getters and setters

def getEmployeeID(self):

    return self.__employeeID


def setEmployeeID(self, employeeID):

    self.__employeeID = employeeID


def __str__(self):

    return "Employee's name is " + self.getFirstName() + " " + self.getLastName()
+ "\n" "Address is " + self.getAddress() + "\n" "Employee ID: " +
str(self.getEmployeeID())
```

```

class Accountant(HRStaff):

    def __init__(self, firstName, lastName, address, employeeID):

        super().__init__(firstName, lastName, address, employeeID)

        # No extra attributes


    def __str__(self):

        return "Employee's name is " + self.getFirstName() + " " + self.getLastName()
+ "\n" "Address is " + self.getAddress() + "\n" "Employee ID: " +
str(self.getEmployeeID())


# Lecturer

lec = PermanentLecturer("Dr. Alice", "Brown", "Frenchay Campus", 1001, "Room
4B12")

print(lec)


# PermanentLecturer

perm_lec = PermanentLecturer("Prof. Cameron", "White", "Frenchay Campus",
102, "Room 5A10")

print(perm_lec)

```

```
# PartTimeLecturer
```

```
part_lec = PartTimeLecturer("Ms. Sarah", "Clark", "Frenchay Campus", 103, 15)
```

```
print(part_lec)
```

```
# Student
```

```
stu = Student("Harry", "Conway", "Frenchay Campus", 2024001)
```

```
print(stu)
```

```
# Undergraduate
```

```
ug = Undergraduate("Raj", "Sharma", "Frenchay Campus", 2024002, 3)
```

```
print(ug)
```

```
# GraduateStudent
```

```
gs = GraduateStudent("Liam", "Scott", "Frenchay Campus", 2024003, "Prof.  
Brown")
```

```
print(gs)
```

```
# HRStaff
```

```
hr = HRStaff("Sophia", "Lee", "Frenchay Campus", "Emp001")
```

```
print(hr)
```

```
# Accountant
```

```
acc = Accountant("Oliver", "Stone", "Frenchay Campus", "Emp002")
```

```
print(acc)
```

```
# Output:
```

```
# Lecturer's name is Dr. Alice Brown
```

```
# Address is Frenchay Campus
```

```
# Lecturer ID: 1001
```

```
# Office Room: Room 4B12
```

```
# Lecturer's name is Prof. Cameron White
```

```
# Address is Frenchay Campus
```

```
# Lecturer ID: 102
```

```
# Office Room: Room 5A10
```

```
# Lecturer's name is Ms. Sarah Clark
```

# Address is Frenchay Campus

# Lecturer ID: 103

# Hours per week: 15

# Student's name is Harry Conway

# Address is Frenchay Campus

# Student ID: 2024001

# Student's name is Raj Sharma

# Address is Frenchay Campus

# Student ID: 2024002

# Year of Study: 3

# Student's name is Liam Scott

# Address is Frenchay Campus

# Student ID: 2024003

# Supervisor's name: Prof. Brown

# Person's name is Sophia Lee

# Address is Frenchay Campus

# Employee ID: Emp001

# Employee's name is Oliver Stone

# Address is Frenchay Campus

# Employee ID: Emp002

## Output obtained in execution:

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS
+ v  1: Python Debug Consc v  [ ] [x] [≡] [⋮] [X]

PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> & 'c:\Users\ASUS\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ASUS\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60644' '--' 'F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9\Person4.py'
Lecturer's name is Dr. Alice Brown
Address is Frenchay Campus
Lecturer ID: 1001
Office Room: Room 4B12
Lecturer's name is Prof. Cameron White
Address is Frenchay Campus
Lecturer ID: 102
Office Room: Room 5A10
Lecturer's name is Ms. Sarah Clark
Address is Frenchay Campus
Lecturer ID: 103
Hours per week: 15
Student's name is Harry Conway
Address is Frenchay Campus
Student ID: 2024001
Student's name is Raj Sharma
Address is Frenchay Campus
Student ID: 2024002
Year of Study: 3
Student's name is Liam Scott
Address is Frenchay Campus
Student ID: 2024003
Supervisor's name: Prof. Brown
Employee's name is Sophia Lee
Address is Frenchay Campus
Employee ID: Emp001
Employee's name is Oliver Stone
Address is Frenchay Campus
Employee ID: Emp002
PS F:\BSc (Hons) Computer Science - Artificial Intelligence Semester III\Modules Handbook\Principles of Programming\Weekly Practical Task\Week 9\practical9> |
```