Week-7: Modules and Inheritance

1. How do you make a module? Give an example of construction of a module using different geometrical shapes and operations on them as its functions.

```
# geometry.py

import math def

square_area(side

):     return side *

side def

square_perimeter

(side):

   return 4 *

side def

circle_area(r

adius):

   return math.pi *

radius**2 def

circle_circumference(r

adius):

   return 2 * math.pi * radius def

triangle_area(base, height):

return 0.5 * base * height def

triangle_perimeter(side1, side2,

side3):

   return side1 + side2 + side3
```

#

main.

py

import

geome

try

side_l

ength

= 5

radius

= 3

base =

4

height

= 6

side

1 =

3

side

2 =

4

side

3 =

5

```
print("Square area:", geometry.square_area(side_length))

print("Square perimeter:",

geometry.square_perimeter(side_length)) print("Circle

area:", geometry.circle_area(radius))

print("Circle circumference:", geometry.circle_circumference(radius))

print("Triangle area:", geometry.triangle_area(base, height))

print("Triangle perimeter:", geometry.triangle_perimeter(side1, side2,
side3))
```

Square area: 25

Square perimeter: 20

Circle area: 28.274333882308138

Circle circumference: 18.84955592153876

Triangle area: 12.0

Triangle perimeter: 12

2. a. Write a function called draw_rectangle that takes a Canvas and a Rectangle as arguments and draws a representation of the Rectangle on the Canvas.

Program:

```
class

Canvas:     def

_init_(self, width,

height):

        self.width = width

self.height = height

self.grid = [[' ' for _ in

range(width)] for _ in

range(height)]
```

```
    def
draw(self):
for row in
self.grid:
print('
'.join(row))


class Rectangle:     def
_init_(self, x, y, width,
height):
        self.x = x
self.y = y
self.width =
width
self.height =
height


def draw_rectangle(canvas, rectangle):
    for row in range(rectangle.y, rectangle.y +
rectangle.height):         for col in range(rectangle.x,
rectangle.x + rectangle.width):
            if row == rectangle.y or row == rectangle.y + rectangle.height - 1
or col == rectangle.x or col == rectangle.x + rectangle.width - 1:
```

```
canvas.grid[row][col] =

'#' canvas = Canvas(10, 6)

rectangle = Rectangle(2,

1, 5, 3)

draw_rectangle(canvas,

rectangle) canvas.draw()
```

Output:

```
  # # # # #

  #       #

  # # # # #
```

b. Add an attribute named color to your Rectangle objects and modify draw_rectangle so that it uses the color attribute as the fill color.

```
import matplotlib.pyplot as

plt import

matplotlib.patches as

patches


class Canvas:


  def __init__(self):

    self.fig, self.ax = plt.subplots()
```

```
    def draw_rectangle(self, rectangle):
        x = rectangle.x
y = rectangle.y
width =
rectangle.width
height =
rectangle.height
color =
rectangle.color
        rect = patches.Rectangle((x, y), width, height, color=color)
self.ax.add_patch(rect)


    def show(self):
plt.axis('equal')
plt.show()


class Rectangle:


    def __init__(self, x, y, width, height, color):
        self.x = x
self.y = y
self.width =
width
self.height =
```

height

self.color =

color

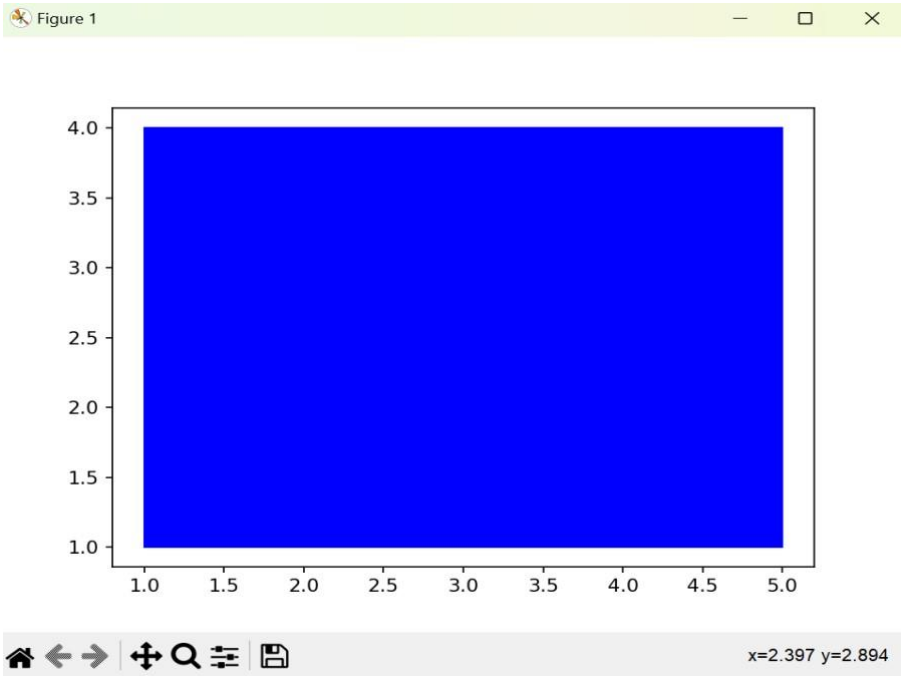
Example usage: canvas =

Canvas() rectangle1 =

Rectangle(1, 1, 4, 3, "blue")

canvas.draw_rectangle(rectang

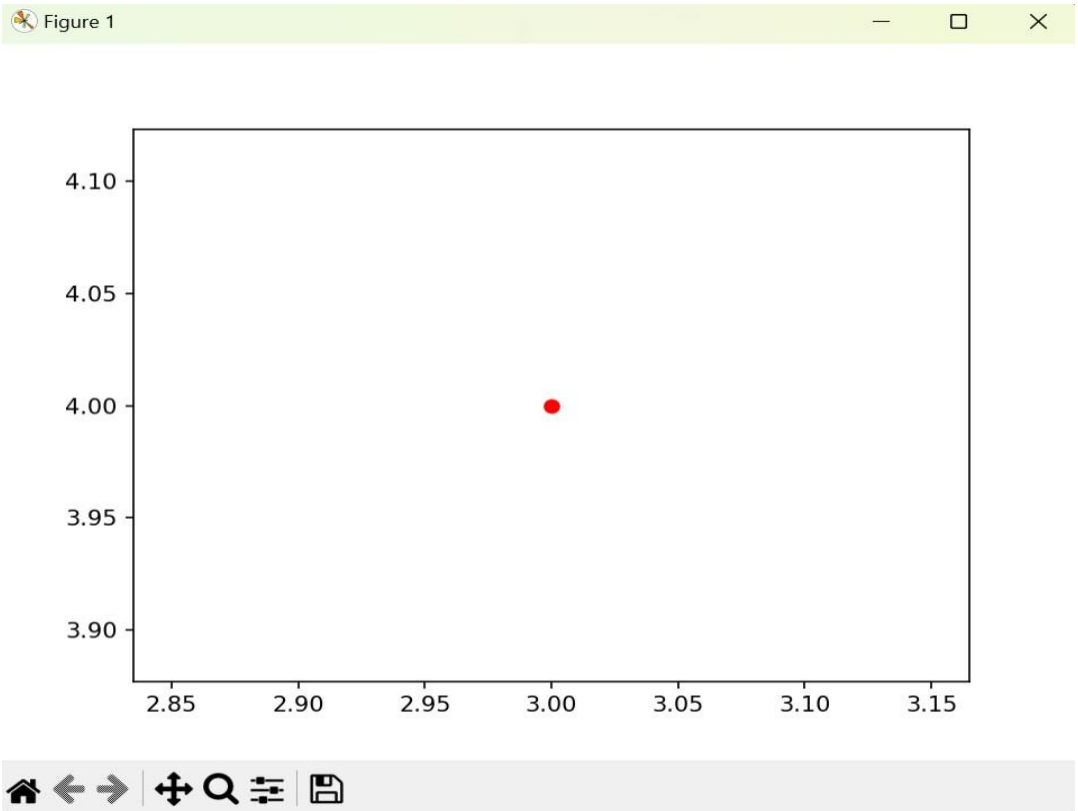
le1) canvas.show()



**c. Write a function called draw_point that takes a Canvas and a Point as arguments and draws a representation of the Point on the Canvas. **

```
import matplotlib.pyplot as
plt import
matplotlib.patches as
patches


class Canvas:


    def __init__(self):
        self.fig, self.ax = plt.subplots()


    def draw_point(self, point):
        x =
point.x
y = point.y
plt.plot(x,
y, 'ro')


    def show(self):
plt.axis('equal')
plt.show()


class Point:


    def __init__(self, x, y):
```

```
        self.x
= x
self.y = y
# Example
usage: canvas =
Canvas() point1
= Point(3, 4)
canvas.draw_po
int(point1)
canvas.show()
```



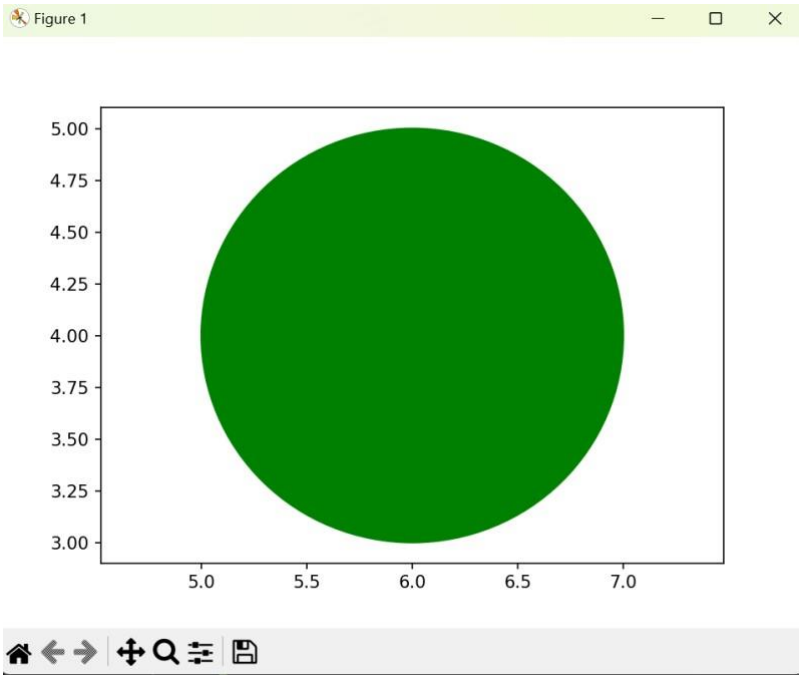
d. Define a new class called Circle with appropriate attributes and instantiate a few Circle objects. Write a function called draw_circle that draws circles on the canvas.

```
import matplotlib.pyplot as
plt import
matplotlib.patches as
patches


class Canvas:


    def __init__(self):
        self.fig, self.ax = plt.subplots()

    def draw_circle(self, circle):
        x =
circle.x         y
= circle.y
radius =
circle.radius
color =
circle.color
        circle = patches.Circle((x, y), radius, color=color)
self.ax.add_patch(circle)


    def show(self):
plt.axis('equal')
plt.show()
```

```
class Circle:


    def __init__(self, x, y, radius, color):
        self.x = x
self.y = y
self.radius = radius
self.color = color #
```

Example usage:

```
canvas = Canvas()
circle1 = Circle(6, 4,
1, "green")
canvas.draw_circle(ci
rcle1) canvas.show()
```



3. Write a Python program to demonstrate the usage of Method Resolution Order (MRO) in multiple levels of Inheritance.

Prog

ram:

```
class
A:
    def show(self):
        print("A class method")


class
B(A):
def
show(s
elf):
        print("B class method")


class
C(A):
def
show(s
elf):
print("
C class
method
")
```

```
class
D(B,
C):
def
show(s
elf):
    print("D class method")


# Instantiate the D class object and call the
show method d_obj = D() d_obj.show()


# Print the Method Resolution Order for class D
print(D.mro())
```

Output:

```
D class method
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>,
<class '__main__.A'>, <class 'object'>]
```