**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(radius):

return math.pi \* radius\*\*2

def circle\_circumference(radius):

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 geometry

side\_length = 5

radius = 3

base = 4

height = 6

side1 = 3

side2 = 4

side3 = 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

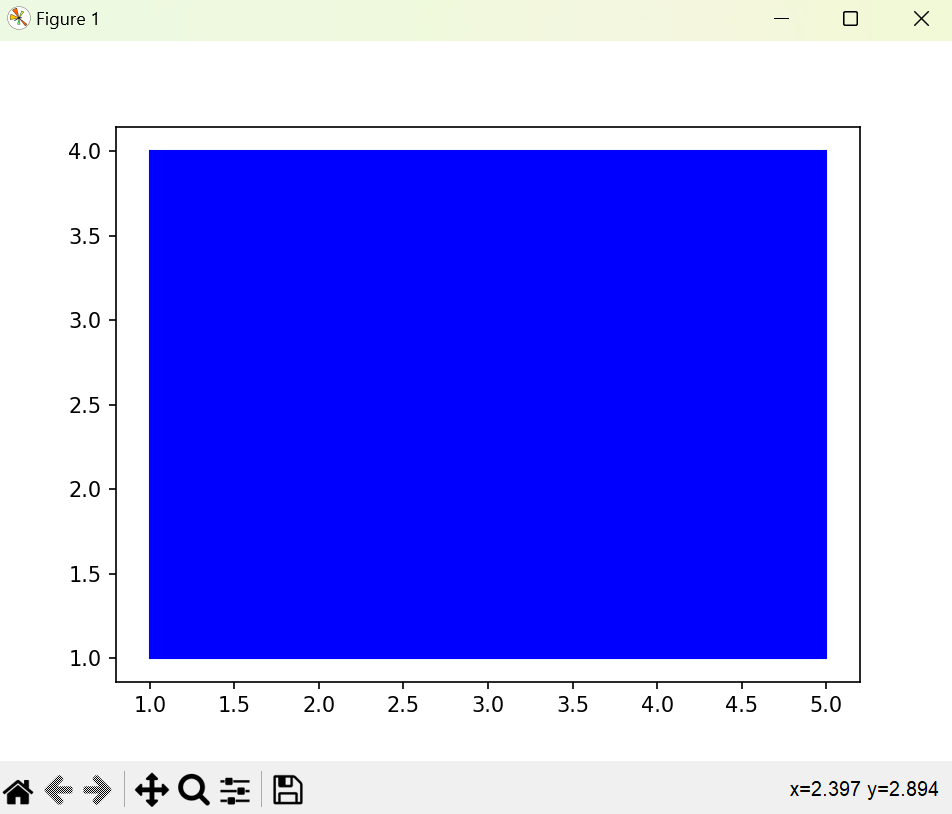
# Example usage:

canvas = Canvas()

rectangle1 = Rectangle(1, 1, 4, 3, "blue")

canvas.draw\_rectangle(rectangle1)

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

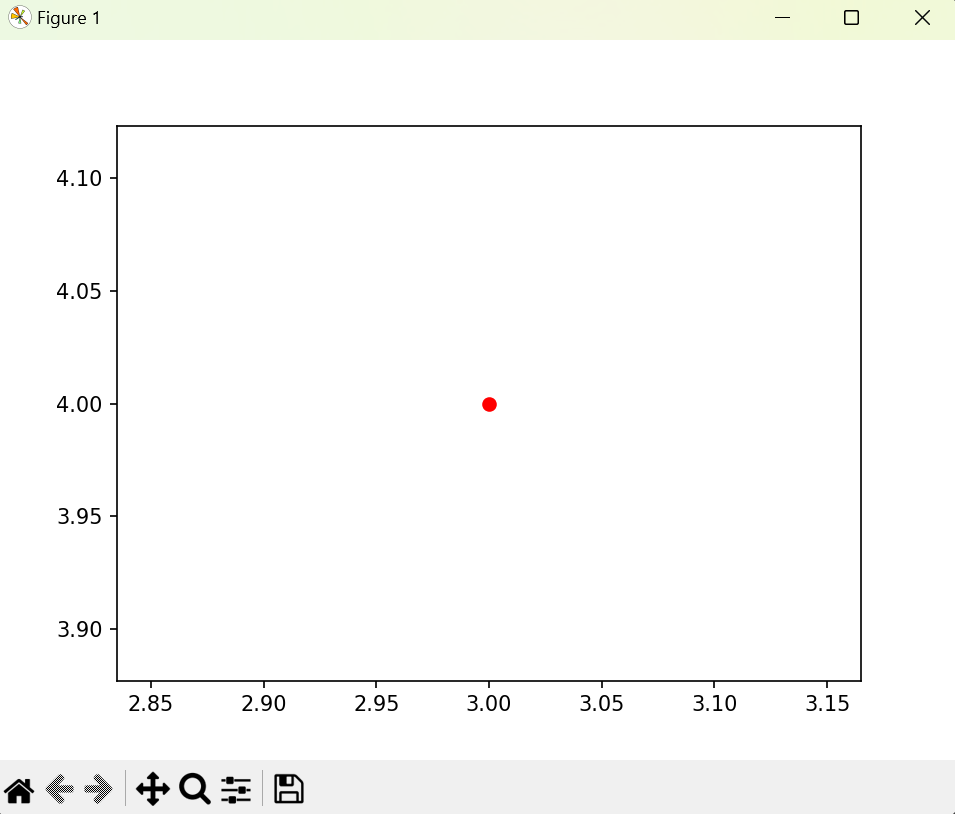
# Example usage:

canvas = Canvas()

point1 = Point(3, 4)

canvas.draw\_point(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

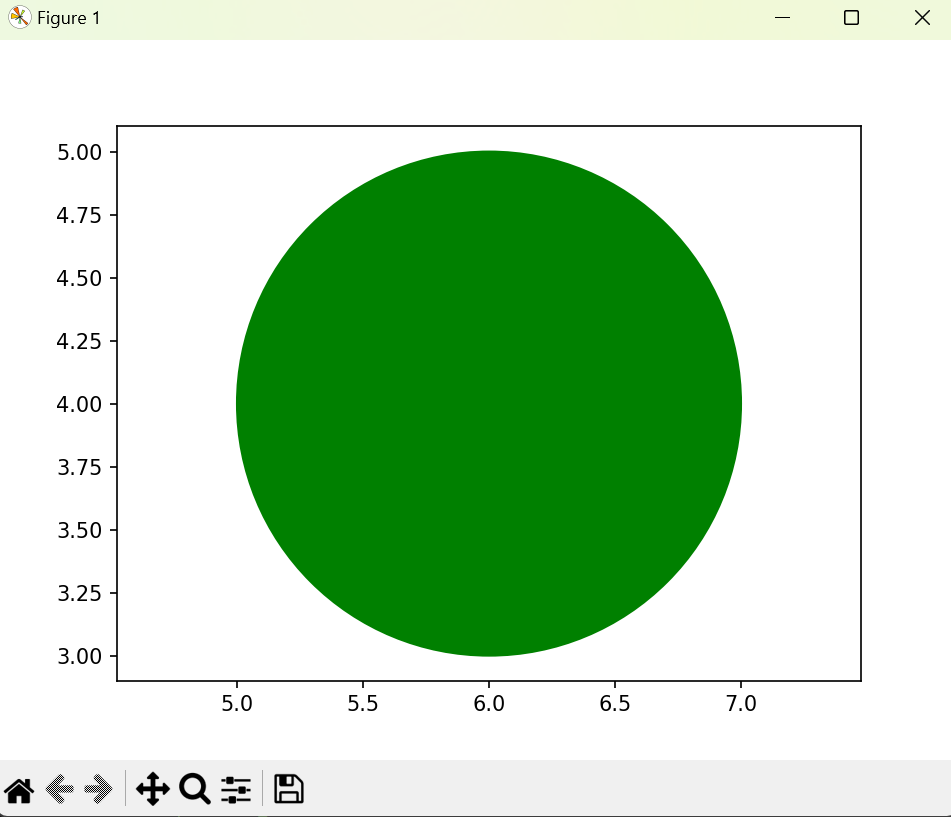
# Example usage:

canvas = Canvas()

circle1 = Circle(6, 4, 1, "green")

canvas.draw\_circle(circle1)

canvas.show()



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

**Program:**

class A:

def show(self):

print("A class method")

class B(A):

def show(self):

print("B class method")

class C(A):

def show(self):

print("C class method")

class D(B, C):

def show(self):

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'>]