A picture containing logo

Description automatically generated

**Homework # 12**

**01286121 Computer Programming**

**Software Engineering Program,**

**Department of Computer Engineering,**

**School of Engineering, KMITL**

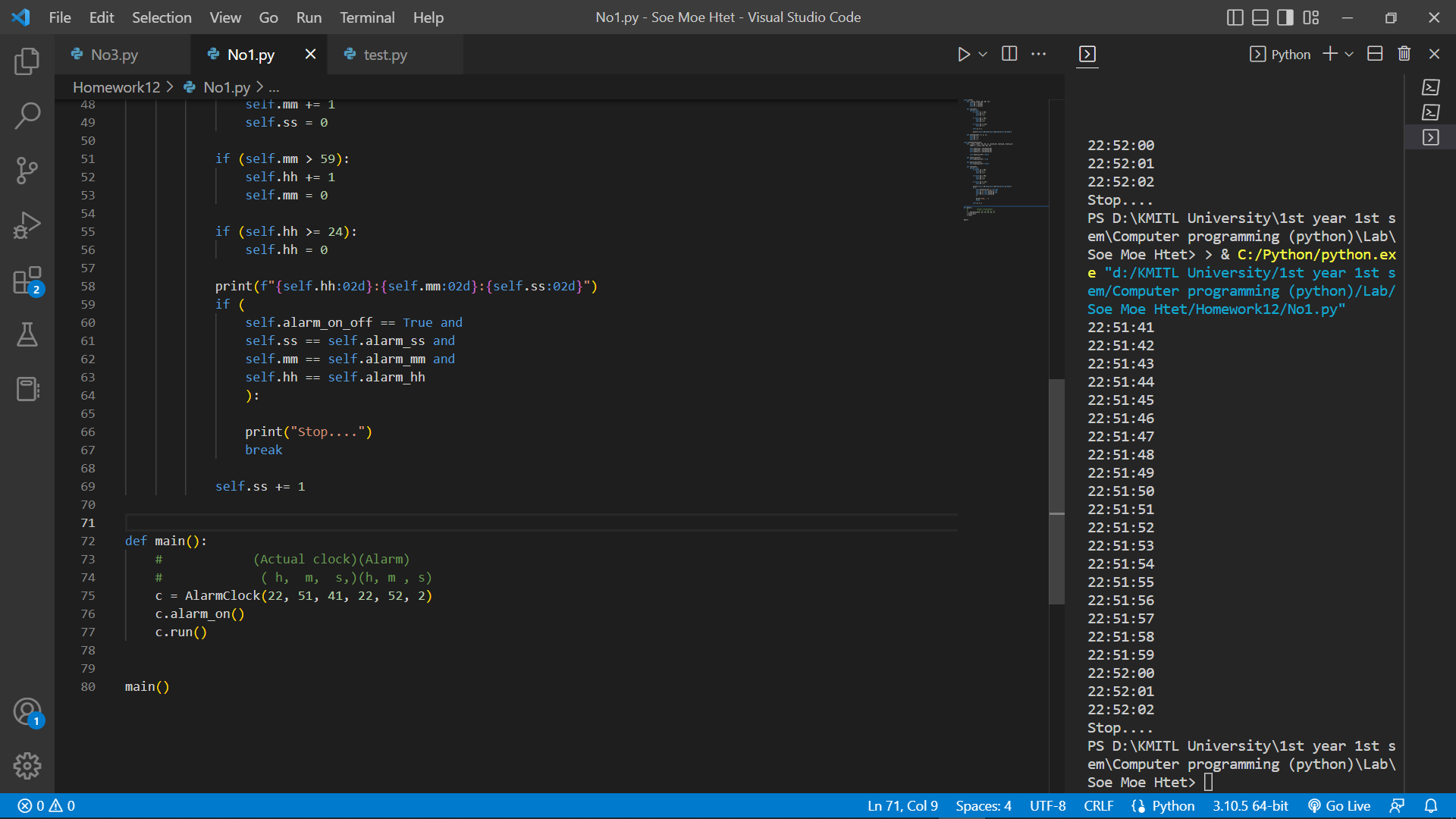
By

65011693 Soe Moe Htet

(Nickname – Stephen)

No1.

Result:



Code:

class Clock:

    def \_\_init\_\_(self, hh, mm, ss):

        self.hh = int(hh)

        self.mm = int(mm)

        self.ss = int(ss)

    def run(self):

        while(True):

            if (self.ss > 59):

                self.mm += 1

                self.ss = 0

            if (self.mm > 59):

                self.hh += 1

                self.mm = 0

            if (self.hh >= 24):

                self.hh = 0

            self.ss += 1

            print(f"{self.hh:02d}:{self.mm:02d}:{self.ss:02d}")

    def setTime(self, h, m, s):

        self.hh = h

        self.mm = m

        self.ss = s

class AlarmClock(Clock):

    def \_\_init\_\_(self, hh, mm, ss, alarm\_hh, alarm\_mm, alarm\_ss):

        super().\_\_init\_\_(hh, mm, ss)

        self.alarm\_hh = int(alarm\_hh)

        self.alarm\_mm = int(alarm\_mm)

        self.alarm\_ss = int(alarm\_ss)

        self.alarm\_on\_off = False

    def alarm\_on(self):

        self.alarm\_on\_off = True

    def alarm\_off(self):

        self.alarm\_on\_off = False

    def run(self):

        while(True):

            if (self.ss > 59):

                self.mm += 1

                self.ss = 0

            if (self.mm > 59):

                self.hh += 1

                self.mm = 0

            if (self.hh >= 24):

                self.hh = 0

            print(f"{self.hh:02d}:{self.mm:02d}:{self.ss:02d}")

            if (

                self.alarm\_on\_off == True and

                self.ss == self.alarm\_ss and

                self.mm == self.alarm\_mm and

                self.hh == self.alarm\_hh

                ):

                print("Stop....")

                break

            self.ss += 1

def main():

    #            (Actual clock)(Alarm)

    #             ( h,  m,  s,)(h, m , s)

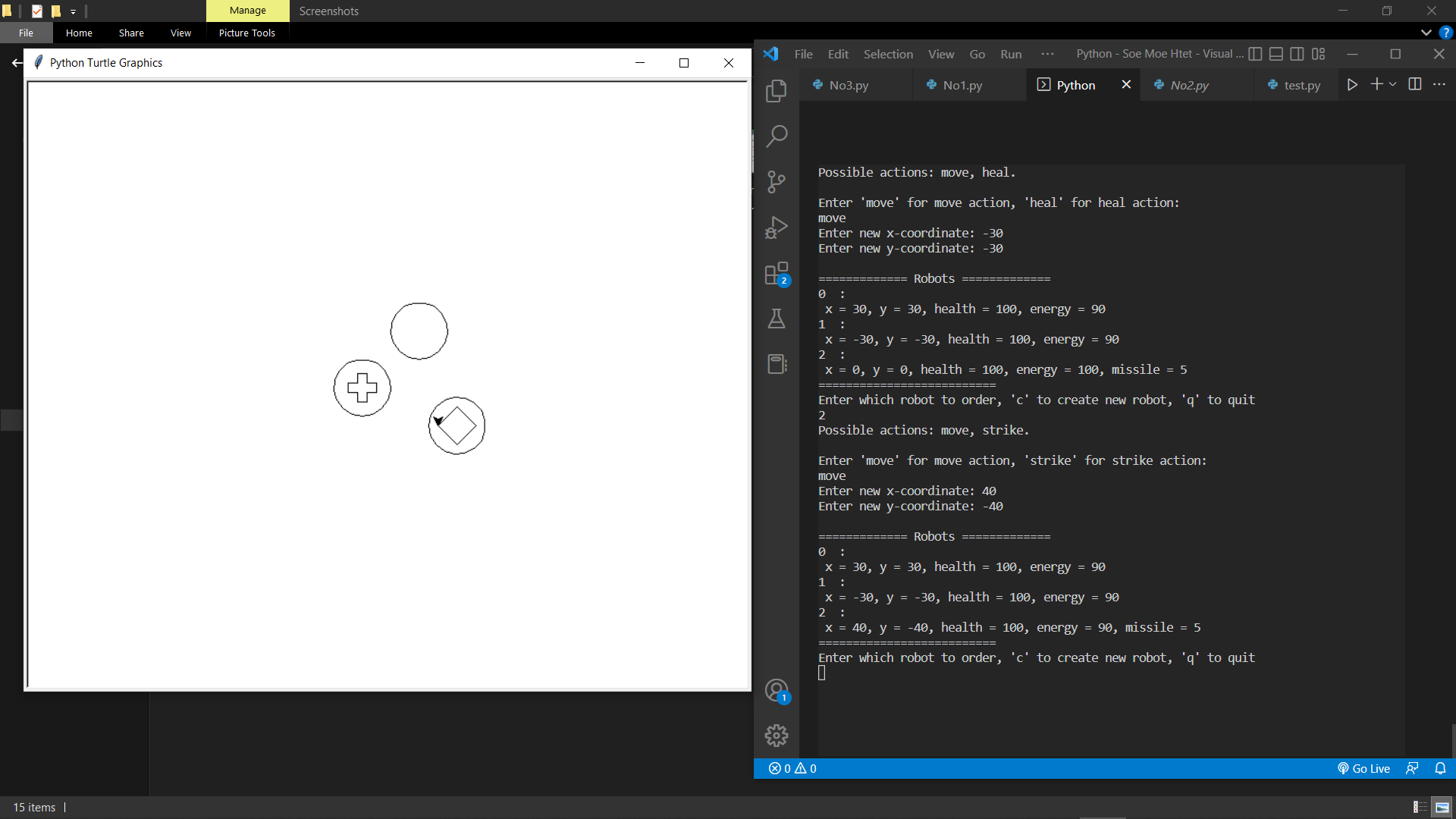
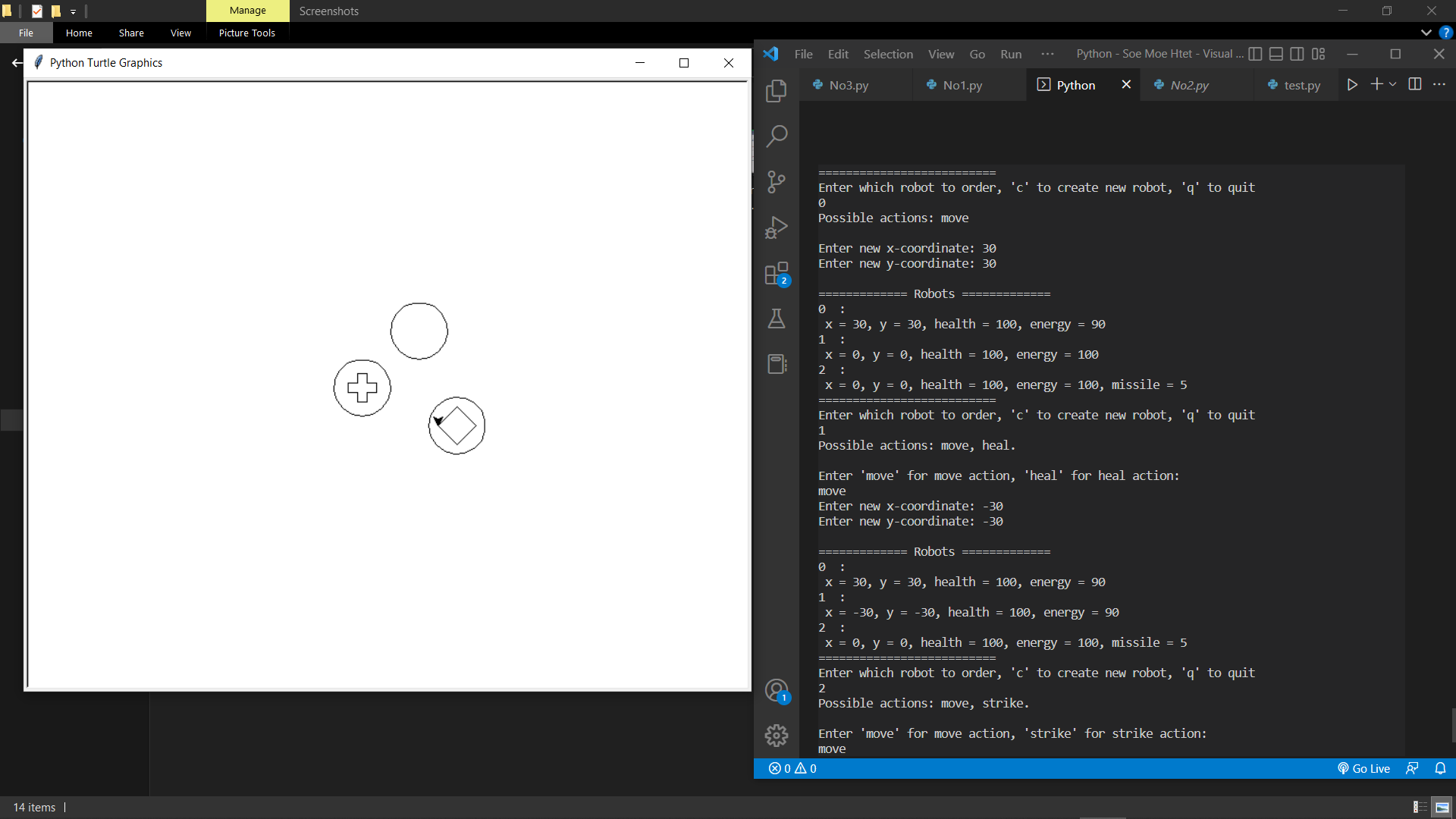
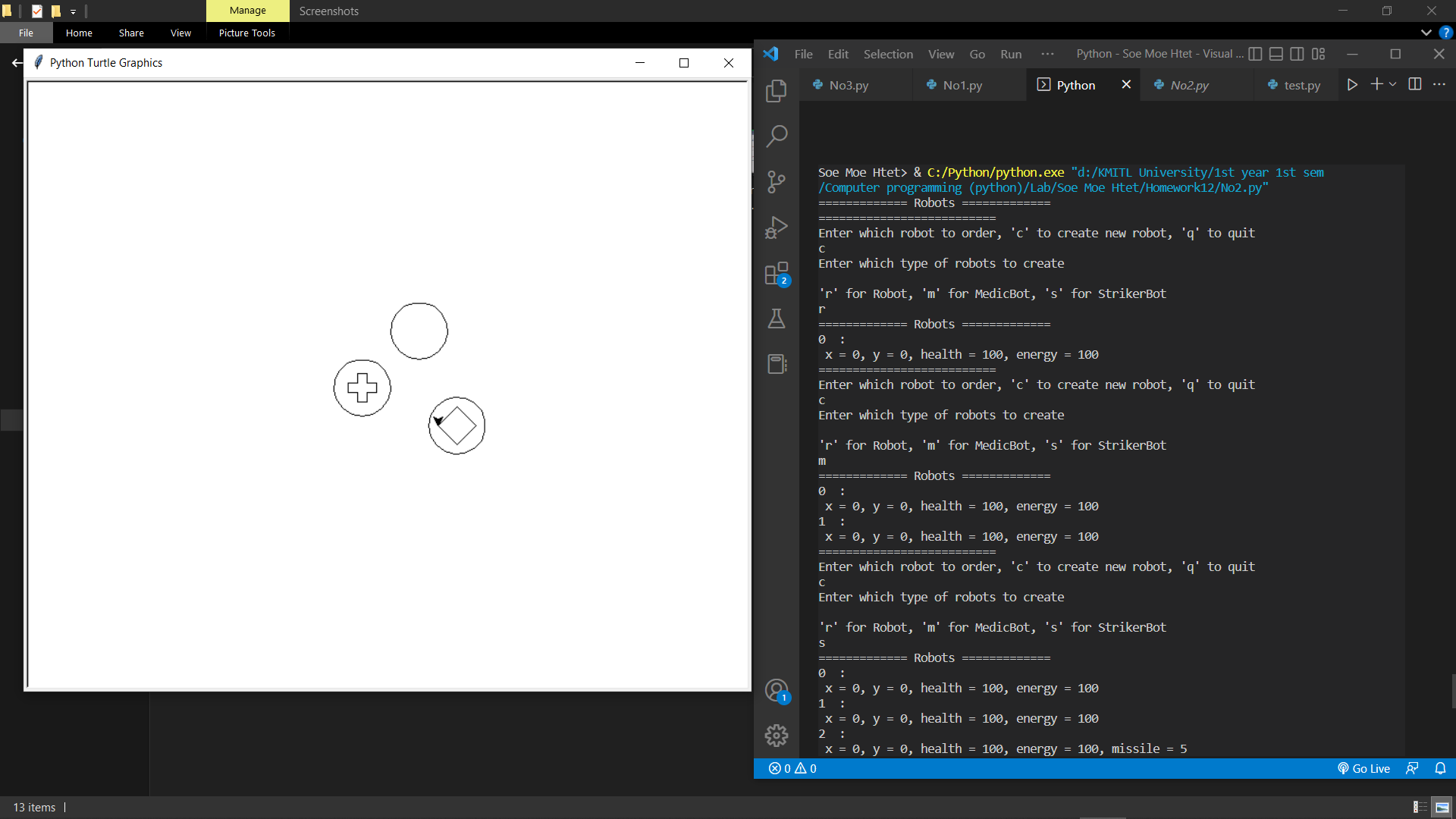
    c = AlarmClock(22, 51, 41, 22, 52, 2)

    c.alarm\_on()

    c.run()

main()

No.2

Result:

Code:

import turtle as t

t.speed(0)

def RobotBattle():

    #robotList stores the list of robots in the battle

    robotList = []

    while True:

        # Clear the screen and draw the robots

        t.clear()

        for robot in robotList:

            robot.draw()

        # Display the status of each robot

        print("============= Robots =============")

        i = 0

        for robot in robotList:

            print(i, " : ")

            robot.displayStatus()

            i += 1

        print("==========================")

        # Ask user which robot to command or to create a new robot

        choice = input("Enter which robot to order, 'c' to create new robot, 'q' to quit \n")

        if choice == "q":

            break

        elif choice == "c":

            print("Enter which type of robots to create\n")

            robotType = input("'r' for Robot, 'm' for MedicBot, 's' for StrikerBot\n")

            if robotType == "r":

                newRobot = Robot()

            elif robotType == "m":

                newRobot = MedicBot()

            elif robotType == "s":

                newRobot = StrikerBot()

            robotList = robotList + [newRobot]

        else:

            n = int(choice)

            robotList[n].command(robotList)

        i = 0

        for robot in robotList:

            if (robot.health <= 0):

                del robotList[i]

            i += 1

class Robot(object):

    def \_\_init\_\_(self):

        self.x = 0

        self.y = 0

        self.health = 100

        self.energy = 100

    def move(self, newX, newY):

        if (self.energy > 0):

            self.x = newX

            self.y = newY

            self.energy -= 10

        elif (self.energy <= 0):

            pass

        print()

    def draw(self):

        t.pu()

        t.setpos(self.x, self.y)

        t.pd()

        t.circle(30)

    def displayStatus(self):

        print(f" x = {self.x}, y = {self.y}, health = {self.health}, energy = {self.energy}")

    def command(self, robotList):

        print("Possible actions: move\n")

        newX = int(input("Enter new x-coordinate: "))

        newY = int(input("Enter new y-coordinate: "))

        self.move(newX, newY)

class MedicBot(Robot):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

    def heal(self, r):

        distancex = self.x - r.x

        distancey = self.y - r.y

        if (self.energy >= 20 and distancex <= 10 and distancey <= 10):

            self.energy -= 20

            r.health += 10

        else:

            pass

    def command(self, robotList):

        print("Possible actions: move, heal.\n")

        command = input("Enter 'move' for move action, 'heal' for heal action: \n")

        if (command == "move"):

            newX = int(input("Enter new x-coordinate: "))

            newY = int(input("Enter new y-coordinate: "))

            self.move(newX, newY)

        elif (command == "heal"):

            robot\_to\_heal = int(input("Choose which robot to heal: "))

            self.heal(robotList[robot\_to\_heal])

    def draw(self):

        super().draw()

        t.penup()

        t.forward(-5)

        t.left(90)

        t.forward(15)

        t.pendown()

        t.right(90)

        t.forward(10)

        for \_ in range(3):

            t.left(90)

            t.forward(10)

            t.right(90)

            t.forward(10)

            t.left(90)

            t.forward(10)

        t.left(90)

        t.forward(10)

        t.right(90)

        t.forward(10)

class StrikerBot(Robot):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.missile = 5

    def strike(self, r):

        distancex = self.x - r.x

        distancey = self.y - r.y

        if (self.energy >= 20 and self.missile > 0 and distancex <= 10 and distancey <= 10):

            self.energy -= 20

            self.missile -= 1

            r.health -= 50

        else:

            pass

    def displayStatus(self):

        print(f" x = {self.x}, y = {self.y}, health = {self.health}, energy = {self.energy}, missile = {self.missile}")

    def command(self, robotList):

        print("Possible actions: move, strike.\n")

        command = input("Enter 'move' for move action, 'strike' for strike action: \n")

        if (command == "move"):

            newX = int(input("Enter new x-coordinate: "))

            newY = int(input("Enter new y-coordinate: "))

            self.move(newX, newY)

        elif (command == "strike"):

            robot\_to\_strike = int(input("Choose which robot to strike: "))

            self.strike(robotList[robot\_to\_strike])

    def draw(self):

        super().draw()

        t.penup()

        t.left(90)

        t.forward(10)

        t.pendown()

        t.right(90)

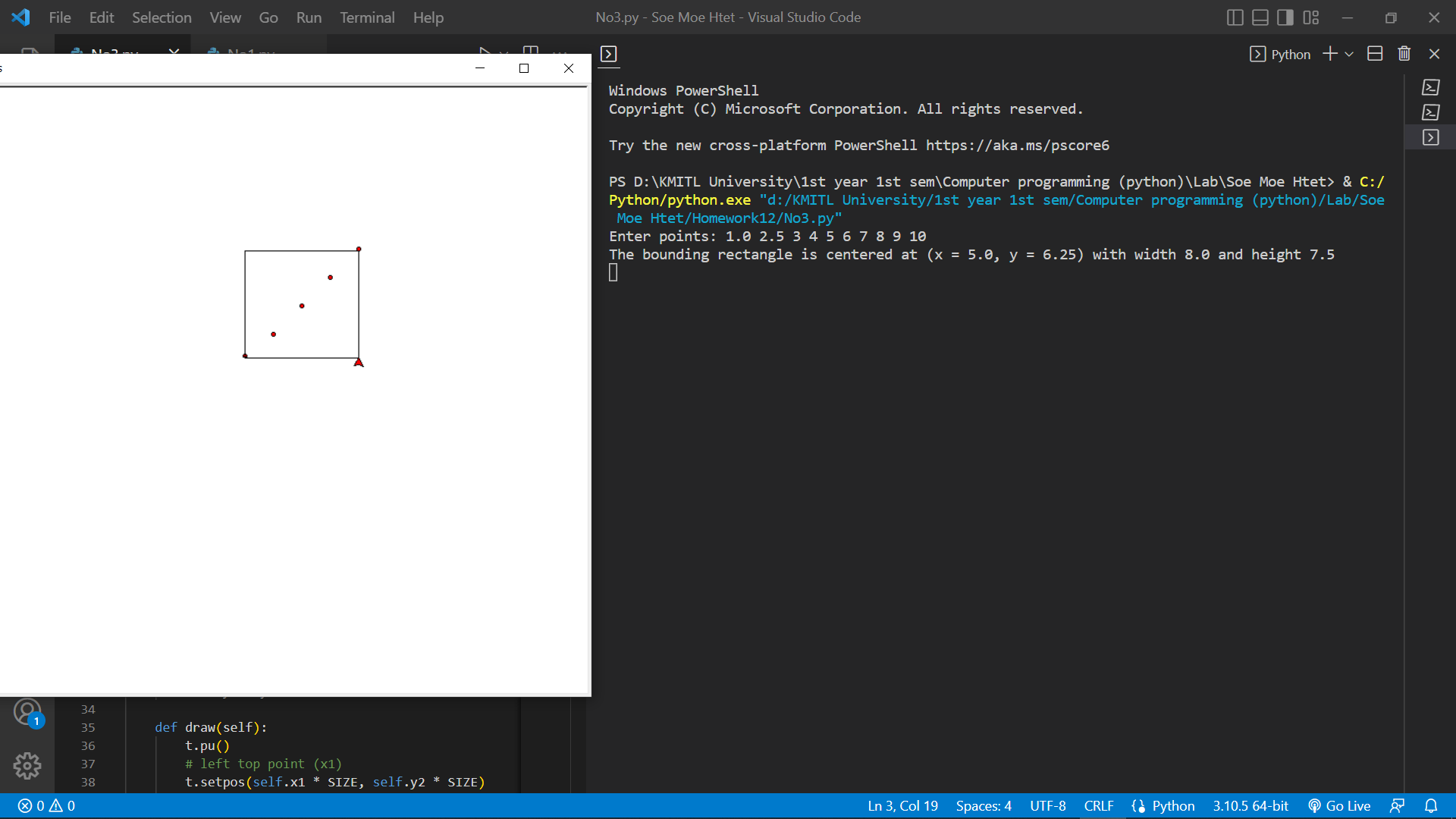
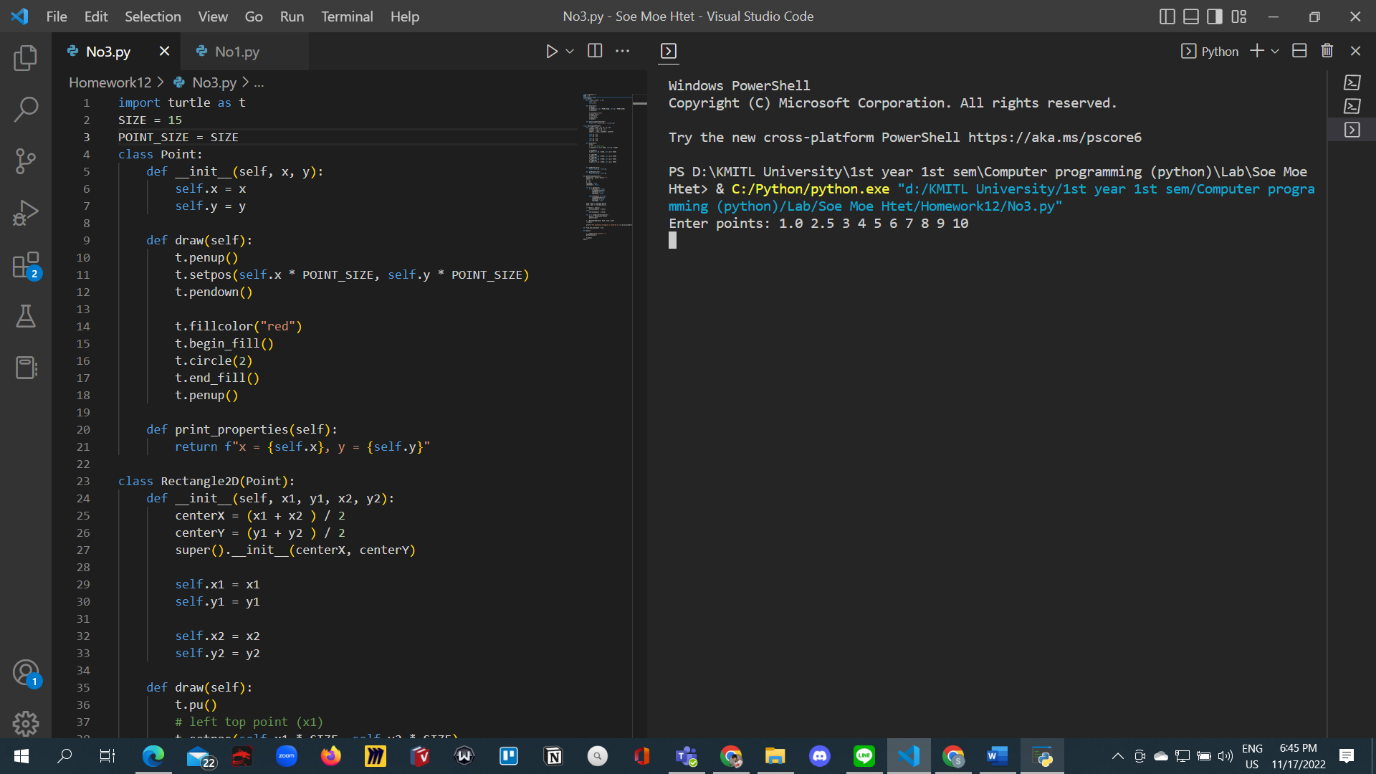
        t.circle(20, 360, 4)

def main():

    RobotBattle()

main()

No.3

Result:

Code:

import turtle as t

SIZE = 15

POINT\_SIZE = SIZE

class Point:

    def \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

    def draw(self):

        t.penup()

        t.setpos(self.x \* POINT\_SIZE, self.y \* POINT\_SIZE)

        t.pendown()

        t.fillcolor("red")

        t.begin\_fill()

        t.circle(2)

        t.end\_fill()

        t.penup()

    def print\_properties(self):

        return f"x = {self.x}, y = {self.y}"

class Rectangle2D(Point):

    def \_\_init\_\_(self, x1, y1, x2, y2):

        centerX = (x1 + x2 ) / 2

        centerY = (y1 + y2 ) / 2

        super().\_\_init\_\_(centerX, centerY)

        self.x1 = x1

        self.y1 = y1

        self.x2 = x2

        self.y2 = y2

    def draw(self):

        t.pu()

        # left top point (x1)

        t.setpos(self.x1 \* SIZE, self.y2 \* SIZE)

        t.pendown()

        t.goto(self.x2 \* SIZE, self.y2 \* SIZE)

        t.right(90)

        t.goto(self.x2 \* SIZE, self.y1 \* SIZE)

        t.right(90)

        t.goto(self.x1 \* SIZE, self.y1 \* SIZE)

        t.right(90)

        t.goto(self.x1 \* SIZE, self.y2 \* SIZE)

    def getWidth(self):

        return self.x1 - self.x2

    def getHeight(self):

        return self.y1 - self.y2

def getRectangle(points):

    pointList = points.split(" ")

    count = 0

    x = []

    y = []

    checkodd = True

    checkeven = False

    for i in pointList:

        if (checkodd == True):

            x.append(float(i))

            checkeven = True

            checkodd = False

        elif(checkeven == True):

            y.append(float(i))

            checkeven = False

            checkodd = True

    minX, maxX = find\_max\_min(x)

    minY, maxY = find\_max\_min(y)

    if len(x) < len(y):

        iterationpoint = len(x)

    else:

        iterationpoint = len(y)

    for i in range(iterationpoint):

        point = Point(x[i], y[i])

        point.draw()

    r = Rectangle2D(maxX, maxY, minX, minY)

    r.draw()

    print(f"The bounding rectangle is centered at ({r.print\_properties()}) with width {r.getWidth()} and height {r.getHeight()}")

def find\_max\_min(point = []):

    min = point[0]

    for i in point:

        if (i < min):

            min = i

    max = point[0]

    for i in point:

        if (i > max):

            max = i

    return min, max

def main():

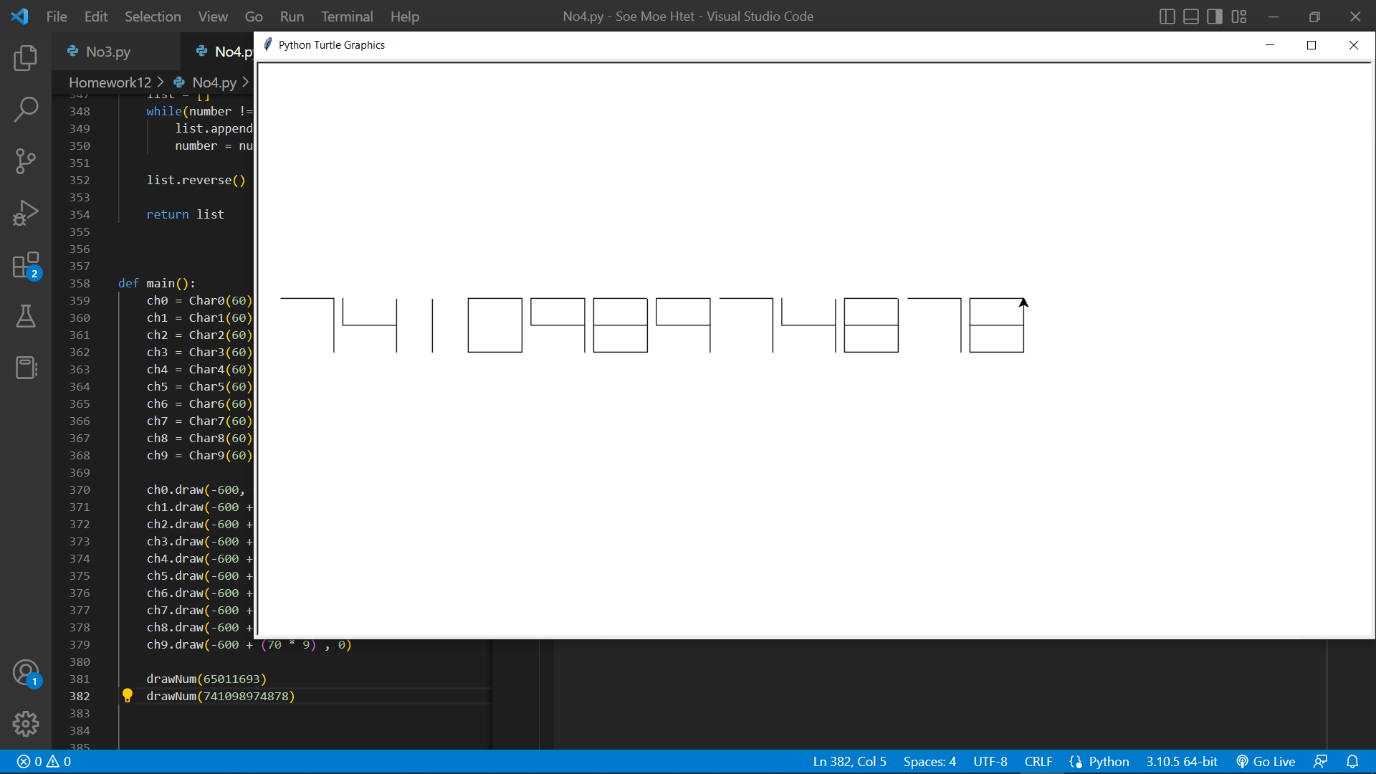
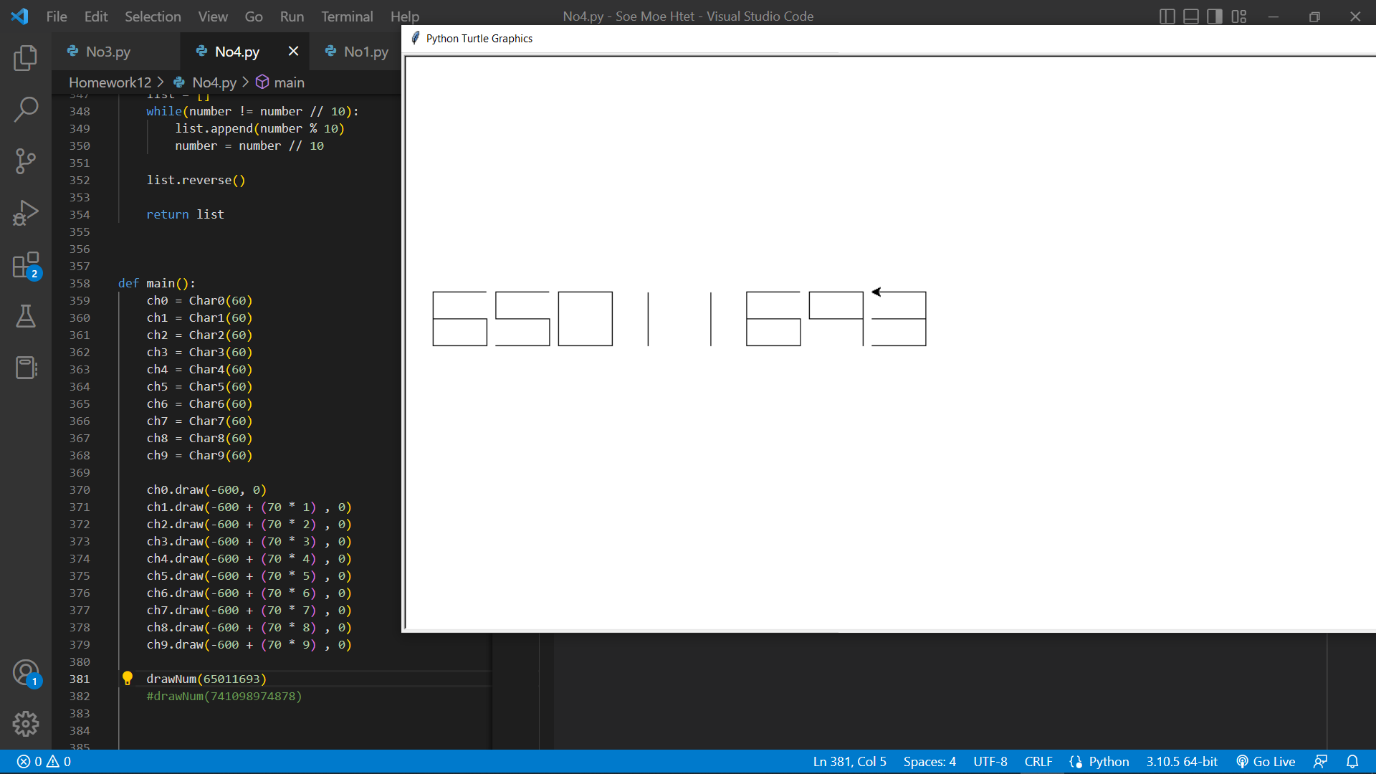
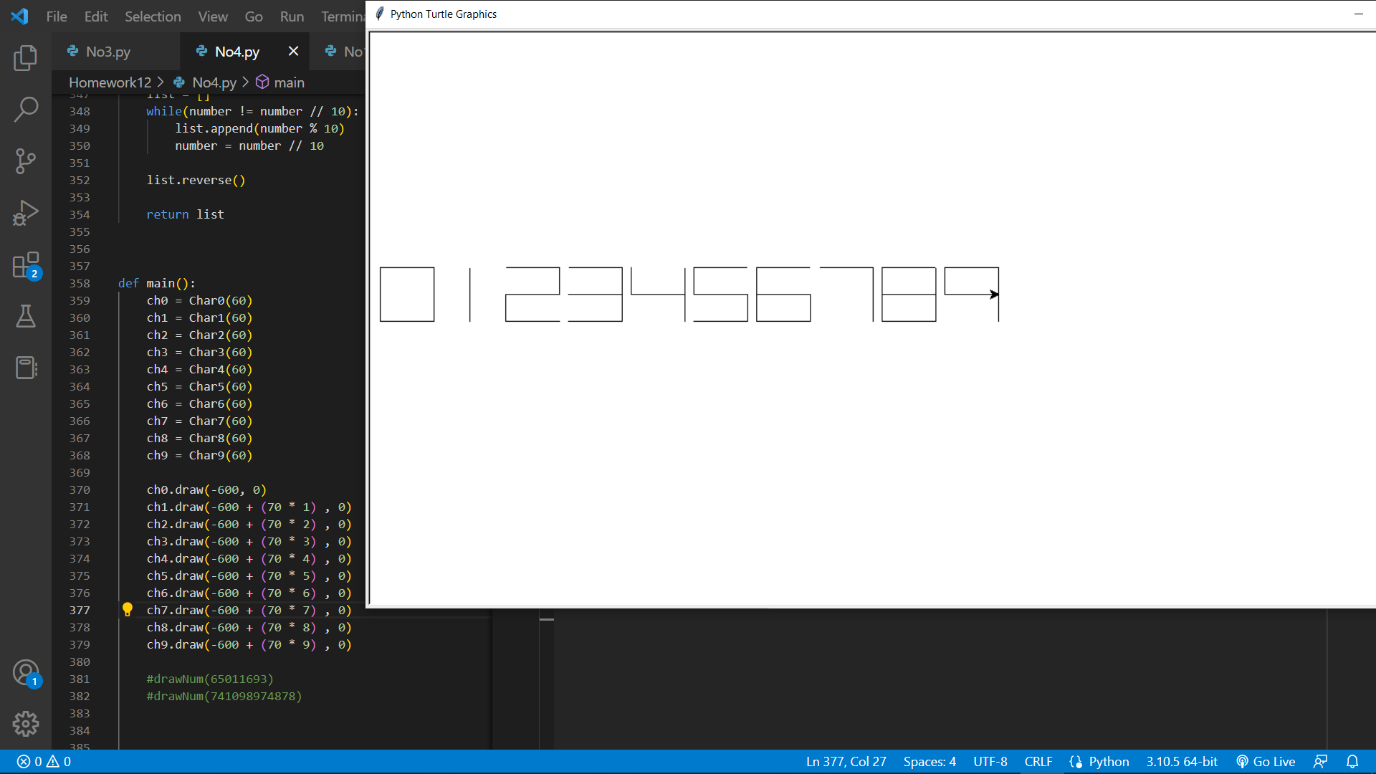
    r = input("Enter points: ")

    getRectangle(r)

    t.done()

main()

No.4 (1.1 + 1.2)

Result:

Code:

from abc import ABC, abstractmethod

import turtle as t

t.speed(0)

class Char(ABC):

    @abstractmethod

    def \_\_init\_\_(self, width):

        self.width = width

    @abstractmethod

    def draw(self, x, y):

        pass

    @abstractmethod

    def getWidth(self):

        return self.width

class Char0(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        t.pendown()

        width = super().getWidth()

        t.seth(0)

        for \_ in range(4):

            t.forward(width)

            t.left(90)

    def getWidth(self):

        return self.width

class Char1(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(0)

        t.forward(width \* 0.5)

        t.pendown()

        t.left(90)

        t.forward(width)

    def getWidth(self):

        return self.width

class Char2(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        t.pendown()

        width = super().getWidth()

        t.right(90)

        t.forward(width)

        t.left(180)

        t.forward(width)

        t.right(90)

        t.forward(width \* 0.5)

        t.right(90)

        t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

    def getWidth(self):

        return self.width

class Char3(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        t.pendown()

        width = super().getWidth()

        t.seth(90)

        t.right(90)

        t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

        t.right(180)

        t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

    def getWidth(self):

        return self.width

class Char4(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(90)

        t.penup()

        t.right(90)

        t.forward(width)

        t.pendown()

        t.left(90)

        t.forward(width)

        t.penup()

        t.left(90)

        t.forward(width)

        t.pendown()

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

    def getWidth(self):

        return self.width

class Char5(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(90)

        t.pendown()

        t.right(90)

        t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

        t.right(90)

        t.forward(width \* 0.5)

        t.right(90)

        t.forward(width)

    def getWidth(self):

        return self.width

class Char6(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(90)

        t.pendown()

        t.right(90)

        t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

        t.right(90)

        t.forward(width \* 0.5)

        t.right(90)

        t.forward(width)

#

        t.penup()

        t.left(180)

        t.forward(width)

#

        t.pendown()

        t.left(90)

        t.forward(width)

    def getWidth(self):

        return self.width

class Char7(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(90)

        t.right(90)

        t.forward(width)

        t.pendown()

        for \_ in range(2):

            t.left(90)

            t.forward(width)

    def getWidth(self):

        return self.width

class Char8(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(90)

        t.pendown()

        t.right(90)

        t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

        t.right(90)

        t.forward(width \* 0.5)

        t.right(90)

        t.forward(width)

#

        t.penup()

        t.left(180)

        t.forward(width)

#

        t.pendown()

        for \_ in range(3):

            t.left(90)

            t.forward(width)

    def getWidth(self):

        return self.width

class Char9(Char):

    def \_\_init\_\_(self, width):

        super().\_\_init\_\_(width)

    def draw(self, x, y):

        t.penup()

        t.setpos(x, y)

        width = super().getWidth()

        t.seth(90)

        t.right(90)

        t.forward(width)

        t.pendown()

        for \_ in range(2):

            t.left(90)

            t.forward(width)

        t.left(90)

        t.forward(width \* 0.5)

        t.left(90)

        t.forward(width)

    def getWidth(self):

        return self.width

def drawNum(x):

    t.clear()

    width = 60

    numberList = return\_list(x)

    print(numberList)

    key\_dict = {0: Char0(width), 1: Char1(width), 2: Char2(width), 3: Char3(width), 4: Char4(width),

           5: Char5(width), 6: Char6(width), 7: Char7(width), 8: Char8(width), 9: Char9(width)}

    X\_coordinate = -600

    Y\_coordinate = 0

    for number in numberList:

        for key in key\_dict:

            if (key == number):

                key\_dict[key].draw(X\_coordinate, Y\_coordinate)

                X\_coordinate += 70

def return\_list(number):

    sum = 0

    list = []

    while(number != number // 10):

        list.append(number % 10)

        number = number // 10

    list.reverse()

    return list

def main():

    ch0 = Char0(60)

    ch1 = Char1(60)

    ch2 = Char2(60)

    ch3 = Char3(60)

    ch4 = Char4(60)

    ch5 = Char5(60)

    ch6 = Char6(60)

    ch7 = Char7(60)

    ch8 = Char8(60)

    ch9 = Char9(60)

    ch0.draw(-600, 0)

    ch1.draw(-600 + (70 \* 1) , 0)

    ch2.draw(-600 + (70 \* 2) , 0)

    ch3.draw(-600 + (70 \* 3) , 0)

    ch4.draw(-600 + (70 \* 4) , 0)

    ch5.draw(-600 + (70 \* 5) , 0)

    ch6.draw(-600 + (70 \* 6) , 0)

    ch7.draw(-600 + (70 \* 7) , 0)

    ch8.draw(-600 + (70 \* 8) , 0)

    ch9.draw(-600 + (70 \* 9) , 0)

    drawNum(65011693)

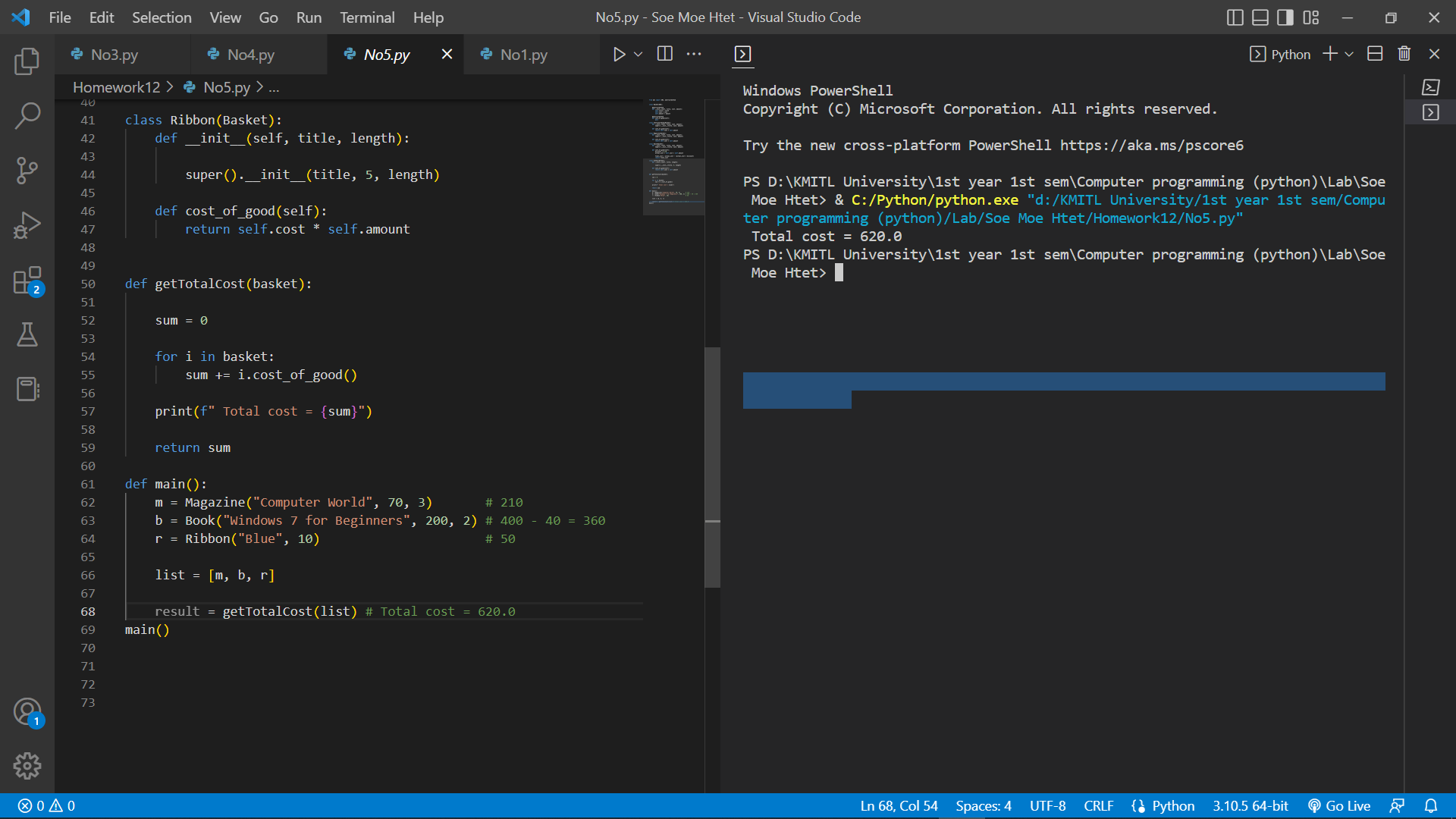
    drawNum(741098974878)

    t.done()

main()

No.5 (2.)

Result:



Code:

from abc import ABC, abstractmethod

class Basket(ABC):

    @abstractmethod

    def \_\_init\_\_(self, title, cost, amount):

        self.title = title

        self.cost = cost

        self.amount = amount

    @abstractmethod

    def cost\_of\_good(self):

        pass

class StationaryGood(Basket):

    def \_\_init\_\_(self, title, cost, amount):

        super().\_\_init\_\_(title, cost, amount)

    def cost\_of\_good(self):

        return self.cost \* self.amount

class Magazine(Basket):

    def \_\_init\_\_(self, title, cost, amount):

        super().\_\_init\_\_(title, cost, amount)

    def cost\_of\_good(self):

        return self.cost \* self.amount

class Book(Basket):

    def \_\_init\_\_(self, title, cost, amount):

        super().\_\_init\_\_(title, cost, amount)

    def cost\_of\_good(self):

        discount = 0.10

        actual\_cost = self.cost \* self.amount

        final\_cost = actual\_cost - (actual\_cost \* discount)

        return final\_cost

class Ribbon(Basket):

    def \_\_init\_\_(self, title, length):

        super().\_\_init\_\_(title, 5, length)

    def cost\_of\_good(self):

        return self.cost \* self.amount

def getTotalCost(basket):

    sum = 0

    for i in basket:

        sum += i.cost\_of\_good()

    print(f" Total cost = {sum}")

    return sum

def main():

    m = Magazine("Computer World", 70, 3)       # 210

    b = Book("Windows 7 for Beginners", 200, 2) # 400 - 40 = 360

    r = Ribbon("Blue", 10)                      # 50

    list = [m, b, r]

    result = getTotalCost(list) # Total cost = 620.0

main()