

**Homework #11**

**01286121 Computer Programming**

**Software Engineering Program,**

**Department of Computer Engineering,**

**School of Engineering, KMITL**

By

68011278 Ananda Stallard

1.

Code:

import time

class Clock:

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

self.hh = hh

self.mm = mm

self.ss = ss

def run(self):

while True:

self.ss += 1

if self.ss >= 60:

self.ss = 0

self.mm += 1

if self.mm >= 60:

self.mm = 0

self.hh += 1

if self.hh >= 24:

self.hh = 0

time.sleep(1)

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

def setTime(self, hh, mm, ss):

self.hh = hh

self.mm = mm

self.ss = ss

class AlarmClock(Clock):

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

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

self.alarm\_hh = alarm\_hh

self.alarm\_mm = alarm\_mm

self.alarm\_ss = alarm\_ss

self.alarm\_on\_off = alarm\_on\_off

def setAlarmTime(self, h, m, s):

self.alarm\_hh = h

self.alarm\_mm = m

self.alarm\_ss = s

def alarm\_on(self):

self.alarm\_on\_off = True

def alarm\_off(self):

self.alarm\_on\_off = False

def run(self):

while True:

self.ss += 1

if self.ss >= 60:

self.ss = 0

self.mm += 1

if self.mm >= 60:

self.mm = 0

self.hh += 1

if self.hh >= 24:

self.hh = 0

if self.hh == self.alarm\_hh and self.mm == self.alarm\_mm and self.ss == self.alarm\_ss and self.alarm\_on\_off == True:

return print("Wakey wakey")

time.sleep(1)

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

c1 = AlarmClock(10, 00, 00, 10, 0, 10, True)

c1.run()

Result:

A black background with white numbers

AI-generated content may be incorrect.

2.

Code:

import turtle

def RobotBattle():

robotList = []

while True:

turtle.clear()

for robot in robotList:

robot.draw()

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

i = 0

for robot in robotList:

print(i, " : ")

robot.displayStatus()

i += 1

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

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

if choice == "q":

break

elif choice == "c":

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

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

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, x, y):

if self.energy > 0:

self.x = x

self.y = y

self.energy -= 10

def draw(self):

turtle.showturtle()

turtle.pu()

turtle.goto(self.x, self.y)

turtle.setheading(0)

turtle.pd()

turtle.circle(25)

turtle.hideturtle()

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")

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):

if self.energy >= 20 and self.x + 20 >= r.x and self.y + 20 >= r.x and self.x - 20 <= r.x and self.y - 20 <= r.y:

self.energy -= 20

r.health += 10

def draw(self):

turtle.showturtle()

turtle.pu()

turtle.goto(self.x, self.y)

turtle.setheading(0)

turtle.pd()

turtle.circle(25)

turtle.left(90)

turtle.pu()

turtle.forward(10)

turtle.right(90)

turtle.forward(5)

turtle.left(90)

turtle.pd()

turtle.forward(10)

for i in range(0, 4):

turtle.right(90)

turtle.forward(10)

turtle.left(90)

turtle.forward(10)

turtle.left(90)

turtle.forward(10)

turtle.hideturtle()

def command(self, robotList):

asking = True

while asking:

choice = input("Possible actions: move, heal")

if choice == "move":

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

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

self.move(newX, newY)

asking = False

elif choice == "heal":

toHeal = int(input("Which Robot to heal (int): "))

if len(robotList) >= toHeal:

self.heal(robotList[toHeal])

asking = False

class StrikerBot(Robot):

def \_\_init\_\_(self, missile = 5):

super().\_\_init\_\_()

self.missile = missile

def strike(self, r):

if self.energy >= 20 and self.missile > 0 and self.x + 10 >= r.x and self.y + 10 >= r.x and self.x - 10 <= r.x and self.y - 10 <= r.y:

self.energy -= 20

self.missile -= 1

r.health -= 50

def displayStatus(self):

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

def draw(self):

turtle.showturtle()

turtle.pu()

turtle.goto(self.x, self.y)

turtle.setheading(0)

turtle.pd()

turtle.circle(25)

turtle.left(90)

turtle.pu()

turtle.forward(10)

turtle.right(45)

turtle.pd()

for i in range(0, 4):

turtle.forward(20)

turtle.left(90)

turtle.left(45)

turtle.hideturtle()

def command(self, robotList):

asking = True

while asking:

choice = input("Possible actions: move, strike")

if choice == "move":

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

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

self.move(newX, newY)

asking = False

elif choice == "strike":

toStrike = int(input("Which Robot to strike: "))

if len(robotList) >= toStrike:

self.strike(robotList[toStrike])

asking = False

turtle.speed(10)

RobotBattle()

Result:

A line art of a game controller

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

3.

Code:

import turtle

class Rectangle2D:

def \_\_init\_\_(self, first\_point):

self.max\_x = first\_point.x

self.min\_x = first\_point.x

self.max\_y = first\_point.y

self.min\_y = first\_point.y

def getRectangle(self, point):

if point.x > self.max\_x:

self.max\_x = point.x

if point.x < self.min\_x:

self.min\_x = point.x

if point.y > self.max\_y:

self.max\_y = point.y

if point.y < self.min\_y:

self.min\_y = point.y

def draw(self):

turtle.pu()

turtle.goto(self.min\_x, self.max\_y)

turtle.pd()

turtle.goto(self.max\_x, self.max\_y)

turtle.goto(self.max\_x, self.min\_y)

turtle.goto(self.min\_x, self.min\_y)

turtle.goto(self.min\_x, self.max\_y)

def info(self):

print(f"The bounding rectangle is centered at ({(self.min\_x + self.max\_x) / 2}, {(self.min\_y + self.max\_y) / 2}) with width {self.max\_x - self.min\_x} and height {self.max\_y - self.min\_y}")

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def draw(self):

turtle.pu()

turtle.goto(self.x, self.y)

turtle.pd()

turtle.dot(1)

values = [float(x) for x in input("Enter points: ").split()]

points = [Point(values[i], values[i + 1]) for i in range(0, len(values), 2)]

for i in points:

i.draw()

bounding = Rectangle2D(points[0])

for i in points[1:]:

bounding.getRectangle(i)

bounding.draw()

turtle.hideturtle()

bounding.info()

turtle.done()

Result:





4.

Code:

import turtle

from abc import ABC, abstractmethod

class Char(ABC):

@abstractmethod

def draw(self, x, y):

pass

@abstractmethod

def getWidth(self):

pass

class Char0(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

for i in range(2):

turtle.pd()

turtle.forward(50)

turtle.right(90)

turtle.forward(100)

turtle.right(90)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char1(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.forward(50)

turtle.right(90)

turtle.forward(100)

turtle.right(90)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char2(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char3(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(180)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char4(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.right(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(180)

turtle.forward(100)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char5(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.forward(50)

turtle.right(180)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char6(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.forward(50)

turtle.right(180)

turtle.forward(50)

turtle.left(90)

turtle.forward(100)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char7(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.left(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(100)

turtle.right(90)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char8(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.left(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

class Char9(Char):

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

super().\_\_init\_\_()

self.start = 0

self.end = 0

def draw(self, x, y):

turtle.showturtle()

turtle.pu()

turtle.setheading(0)

turtle.goto(x, y)

self.start = turtle.xcor()

turtle.pd()

turtle.left(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.left(180)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

self.end = turtle.xcor()

turtle.hideturtle()

def getWidth(self):

return print(round(self.end - self.start))

zero = Char2()

zero.draw(0, 0)

zero.getWidth()

turtle.done()

Result:

A black line on a white background

AI-generated content may be incorrect. A black line drawing of a square

AI-generated content may be incorrect.

 

5.

Code:

from abc import ABC, abstractclassmethod

class StationaryGood(ABC):

@abstractclassmethod

def getCost(self):

pass

class Magazine(StationaryGood):

def \_\_init\_\_(self, title, price, quantity):

self.title = title

self.price = price

self.quantity = quantity

def getCost(self):

return self.price \* self.quantity

class Book(StationaryGood):

def \_\_init\_\_(self, title, price, quantity):

self.title = title

self.price = price

self.quantity = quantity

def getCost(self):

discount\_price = self.price \* 0.9

return discount\_price \* self.quantity

class Ribbon(StationaryGood):

def \_\_init\_\_(self, color, length\_m):

self.color = color

self.length\_m = length\_m

def getCost(self):

return 5 \* self.length\_m

def getTotalCost(basket):

total = 0

for item in basket:

total += item.getCost()

return total

item1 = Magazine("Computer World", 70, 3)

item2 = Book("Windows 7 for Beginners", 200, 2)

item3 = Ribbon("Blue", 10)

items = [item1, item2, item3]

total = getTotalCost(items)

print(f"Total cost of goods: {total:.2f} Baht")

Result:

