**Assignment 11**

Part 1

Design a class called “Rectangle” which has the following properties:

* A constructor that accepts 4 arguments - a width, a length, and x position and a y position. The class should store these 4 values as instance variables.
* A method called “get\_area”. This method should accept no arguments and return the area of the rectangle described using the formula length x width
* A method called “get\_perimeter”. This method should accept no arguments and return the perimeter of the rectangle described using the formula 2 x length + 2 x width

Next, construct two rectangles using your newly created class using the following information:

* Rectangle #1: width of 10, length of 15, position of (5, 3)
* Rectangle #2: width of 3, length of 5, position of (15, 7)

Finally, write a main program that refers to these rectangles and accesses their properties & methods to generate the following output. No user input is required - simply create two rectangles using the above information and call the appropriate methods to generate the output below.

Rectangle #1

\* Coordinates: (5, 3)

\* Area: 150

\* Perimeter: 50

Rectangle #2

\* Coordinates: (15, 7)

\* Area: 15

\* Perimeter: 16

This program should be named as follows: LastNameFirstName\_assign11\_part1.py

Part 2

In this program you will be writing a **class** to simulate a “gumball machine” that you would see at a store. Your class should work as follows:

**Constructor**

* The constructor should accept a capacity for the gumball machine (how many gumballs it is filled with - an integer). The gumball machine should store this capacity as an instance variable. No data validation is required.
* The constructor should also store an instance variable to keep track of how much money is in the machine. All machines are constructed to be empty and have no money in them.
* The constructor should also create a new instance variable (a list) to hold that many gumballs. Fill this list with a random set of gumballs - each gumball can be either red, green or blue. For example, if you create a gumball machine with a capacity of 4 the machine could create a list that looks like the following: [‘red’, ‘green’, ‘green’, ‘blue’]
* The constructor should ‘announce’ that it was constructed with the desired capacity

**Methods**

* report: this method should accept no arguments and simply report out the current status of the gumball machine. For example:

Gumball Machine Report:

\* Gumballs in machine: 5

\* Money in machine: $0.00

* dispense: this method should accept an argument - a coin value - and determine if that coin value is a quarter (i.e. 0.25). If so, a gumball should be removed from the the list and reported to the user. The machine should also accept the coin and increase its internal count of how much money is in the machine. Note that if the gumball machine is empty a new gumball will not be dispensed, and the coin should be rejected.
* count\_gumballs\_by\_type: this method should accept a single argument - a string representing the type of gumball - and print out how many types of that gumball are left in the machine.

Here is some sample code you can run to test your program, along with a possible set of output (which will be different since your gumball machine will contain a random assortment of gumballs):

# EXPECTED OUTPUT

Gumball Machine created with 5 random gumballs

Gumball Machine Report:

\* Gumballs in machine: 5

\* Money in machine: $0.00

There are 1 gumballs of type red in the machine

There are 1 gumballs of type green in the machine

There are 3 gumballs of type blue in the machine

Invalid coin, no gumball will be dispensed

Invalid coin, no gumball will be dispensed

Invalid coin, no gumball will be dispensed

Gumball Machine Report:

\* Gumballs in machine: 5

\* Money in machine: $0.00

There are 1 gumballs of type red in the machine

There are 1 gumballs of type green in the machine

There are 3 gumballs of type blue in the machine

Accepting 0.25; Dispensing a red gumball

Accepting 0.25; Dispensing a blue gumball

Accepting 0.25; Dispensing a blue gumball

Gumball Machine Report:

\* Gumballs in machine: 2

\* Money in machine: $0.75

There are 0 gumballs of type red in the machine

There are 1 gumballs of type green in the machine

There are 1 gumballs of type blue in the machine

Accepting 0.25; Dispensing a green gumball

Accepting 0.25; Dispensing a blue gumball

Machine is empty, no gumball will be dispensed

Gumball Machine Report:

\* Gumballs in machine: 0

\* Money in machine: $1.25

There are 0 gumballs of type red in the machine

There are 0 gumballs of type green in the machine

There are 0 gumballs of type blue in the machine

# TESTER CODE

machine = Gumball\_Machine(5)

machine.report()

machine.count\_gumballs\_by\_type("red")

machine.count\_gumballs\_by\_type("green")

machine.count\_gumballs\_by\_type("blue")

machine.dispense(0.10)

machine.dispense(0.50)

machine.dispense(0.01)

machine.report()

machine.count\_gumballs\_by\_type("red")

machine.count\_gumballs\_by\_type("green")

machine.count\_gumballs\_by\_type("blue")

machine.dispense(0.25)

machine.dispense(0.25)

machine.dispense(0.25)

machine.report()

machine.count\_gumballs\_by\_type("red")

machine.count\_gumballs\_by\_type("green")

machine.count\_gumballs\_by\_type("blue")

machine.dispense(0.25)

machine.dispense(0.25)

machine.dispense(0.25)

machine.report()

machine.count\_gumballs\_by\_type("red")

machine.count\_gumballs\_by\_type("green")

machine.count\_gumballs\_by\_type("blue")

This program should be named as follows: LastNameFirstName\_assign11\_part2.py

**Part 3** You’ve been hired by a large telecommunications company to write a program to help people add and remove apps from their smartphone. To do this you should write a CLASS that models a smartphone. Your class should do the following:

class Smartphone:

# construct a new Smartphone

# smartphones need to keep track of how much space they have left (integer)

# they also need to keep track of their name (string)

# smartphones will need some kind of internal system to keep track of all of

# the apps that are installed, along with their size. a list or a dictionary

# would be useful here.

# when a phone is constructed the 'report' method should be called (see below)

# this method returns nothing and simply prints the desired output to the user

def \_\_init\_\_(self, capacity, name):

# add a new app to the smartphone given an appname (string) and an appsize (integer)

# if the app is already installed, reject it. if the phone cannot hold any additional

# apps because the capacity has been reached, reject it.

# this method returns nothing and simply prints the desired output to the user

def add\_app(self, appname, appsize):

# removes an app from the phone based on appname (string)

# if the app is not installed, reject it

# this method returns nothing and simply prints the desired output to the user

def remove\_app(self, appname):

# checks to see if an app is installed based on appname (string)

# returns True if the app is installed, False if it is not

def has\_app(self, appname):

# returns the current space available on the phone (integer)

def get\_available\_space(self):

# prints a detailed report that describes the following:

# Name of phone

# Capacity of phone

# Available space

# # of apps installed

# a listing of all apps installed, in alphabetical order, with their sizes

# this method returns nothing and simply prints the desired output to the user

def report(self):

Test your class and make sure it works as you expect (you will need to write your own tester program for this). Next, write a program that asks the user to create a new phone and then allows them to use all of the features in your class. A sample running of this program is below:

Size of your new smartphone (32, 64 or 128 GB): 64

Smartphone name: Craig's iPhone

Smartphone created!

Name: Craig's iPhone

Capacity: 0 out of 64 GB

Available space: 64

Apps installed: 0

(r)eport, (a)dd app, r(e)move app or (q)uit: a

App name to add: Angry Birds

App size in GB: 10

(r)eport, (a)dd app, r(e)move app or (q)uit: r

Name: Craig's iPhone

Capacity: 10 out of 64 GB

Available space: 54

Apps installed: 1

\* Angry Birds is using 10 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: a

App name to add: Candy Crush

App size in GB: 15

(r)eport, (a)dd app, r(e)move app or (q)uit: a

App name to add: Facebook

App size in GB: 10

(r)eport, (a)dd app, r(e)move app or (q)uit: r

Name: Craig's iPhone

Capacity: 35 out of 64 GB

Available space: 29

Apps installed: 3

\* Angry Birds is using 10 GB

\* Candy Crush is using 15 GB

\* Facebook is using 10 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: a

App name to add: Instagram

App size in GB: 15

(r)eport, (a)dd app, r(e)move app or (q)uit: a

App name to add: TikTok

App size in GB: 15

Cannot install app, no available space

(r)eport, (a)dd app, r(e)move app or (q)uit: r

Name: Craig's iPhone

Capacity: 50 out of 64 GB

Available space: 14

Apps installed: 4

\* Angry Birds is using 10 GB

\* Candy Crush is using 15 GB

\* Facebook is using 10 GB

\* Instagram is using 15 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: e

App name to remove: Facebook

App removed: Facebook

(r)eport, (a)dd app, r(e)move app or (q)uit: a

App name to add: TikTok

App size in GB: 15

(r)eport, (a)dd app, r(e)move app or (q)uit: r

Name: Craig's iPhone

Capacity: 55 out of 64 GB

Available space: 9

Apps installed: 4

\* Angry Birds is using 10 GB

\* Candy Crush is using 15 GB

\* Instagram is using 15 GB

\* TikTok is using 15 GB

(r)eport, (a)dd app, r(e)move app or (q)uit: q

Goodbye!

This program should be named as follows: LastNameFirstName\_assign11\_part3.py