PHYS2160 Introductory Computational Physics
2021/22 Solutions to Exercise 2

1. (a)
```
# sphere.py
# This program defines the class Sphere for representing spheres.
# Written on 24 Feb 2020 by F K Chow

import math

class Sphere:
    """ A class representing spheres """

    def __init__(self, radius):
        """ Initialize the sphere with a radius """
        self.radius = radius

    def getRadius(self):
        """ Return the radius of the sphere """
        return self.radius

    def surfaceArea(self):
        """ Return the surface area of the sphere """
        return 4*math.pi*self.radius**2

    def volume(self):
        """ Return the volume of the sphere """
        return 4*math.pi*self.radius**3/3
```

   (b)
```
# quadratic.py
# This program defines the class Quadratic for representing the
# quadratic function f(x) = ax^2 + bx + c.
# Last update on 13 Jan 2021 by F K Chow

import cmath
import math
import numpy as np

class Quadratic:
    """ A class representing the quadratic function f(x) = ax^2 + bx
        + c """

    def __init__(self, a, b, c):
        """ Initialize f(x) with its coefficients a, b, c """
        self.a = a
        self.b = b
        self.c = c

    def value(self, x):
        """ Return the value of f at the point x """
        return self.a*x**2+self.b*x+self.c
```

```
    def table(self, n, L, R):
        """ Print a table of x and f values at n values of x in the
            interval [L,R] """
        print("{:>8s} {:>8s}".format("x","f(x)"))
        for x in np.linspace(L, R, n):
            print("{:8.3f} {:8.3f}".format(x,self.value(x)))

    def roots(self):
        """ Return the roots of f(x) = 0 """
        dis = (self.b)**2-4*(self.a)*(self.c)
        denom = 2*self.a
        if abs(dis) < 1e-8:
            return -self.b/denom, -self.b/denom
        elif dis > 0:
            sqrtdis = math.sqrt(dis)
            return (-self.b+sqrtdis)/denom, (-self.b-sqrtdis)/denom
        else:
            sqrtdis = cmath.sqrt(dis)
            return (-self.b+sqrtdis)/denom, (-self.b-sqrtdis)/denom
```

2. (a)
```
# card.py
# This program defines the class Card for representing a playing card.
# Last update on 25 Feb 2022 by F K Chow

class Card:
    """ A class representing a playing card """

    def __init__(self, rank, suit):
        """ Initialize the card with its rank and suit """
        self.rank = rank
        self.suit = suit

    def getRank(self):
        """ Return the rank of the card """
        return self.rank

    def getSuit(self):
        """ Return the suit of the card """
        return self.suit

    def value(self):
        """ Return the Blackjack value of the card """
        if self.rank > 10:
            return 10
        else:
            return self.rank

    def __str__(self):
```

```
        """ Return a string that names the card """
        name = ""
        if self.rank == 1:
            name += "Ace"
        elif self.rank == 11:
            name += "Jack"
        elif self.rank == 12:
            name += "Queen"
        elif self.rank == 13:
            name += "King"
        else:
            name += str(self.rank)
        name += " of "
        if self.suit == "d":
            name += "Diamonds"
        elif self.suit == "c":
            name += "Clubs"
        elif self.suit == "h":
            name += "Hearts"
        else:
            name += "Spades"
        return name

    def __repr__(self):
        """ Return a string such that eval applied to the string
            recreates the instance """
        return "Card({:s}, '{:s}')".format(str(self.rank),
                                           str(self.suit))
```

(b)
```
# rationalnumber.py
# This program defines the class RationalNumber for representing
# rational numbers.
# Last update on 13 Jan 2021 by F K Chow

def gcd(a,b):
    """ Return the greatest common divisor of a and b """
    # Everything divides 0
    if a == 0:
        return b
    if b == 0:
        return a
    # base case
    if a == b:
        return a
    # a is greater
    if (a > b):
        return gcd(a-b, b)
    return gcd(a, b-a)
```

```
class RationalNumber:
    """ A class representing rational numbers """

    def __init__(self, num, den):
        """ Initialize the rational number with its numerator and """
            denominator """
        g = gcd(abs(num), abs(den))
        self.num = int(num/g)
        self.den = int(den/g)

    def __add__(self, other):
        """ Return the sum of two rational numbers as a RationalNumber
            object """
        num = self.num*other.den + other.num*self.den
        den = self.den*other.den
        return RationalNumber(num, den)

    def __sub__(self, other):
        """ Return the difference of two rational numbers as a
            RationalNumber object """
        num = self.num*other.den - other.num*self.den
        den = self.den*other.den
        return RationalNumber(num, den)

    def __mul__(self, other):
        """ Return the product of two rational numbers as a
            RationalNumber object """
        return RationalNumber(self.num*other.num, self.den*other.den)

    def __truediv__(self, other):
        """ Return the quotient of two rational numbers as a
            RationalNumber object """
        return RationalNumber(self.num*other.den, self.den*other.num)

    def display(self):
        """ Print the fraction in the form numerator/denominator """
        print("{:s}/{:s}".format(str(self.num), str(self.den)))
```

3.
```
# employee.py
# This program defines the class Employee for storing the data of the
# employee in a company.
# Last update on 13 Jan 2021 by F K Chow

class Employee:
    """ A class storing the data of the employee in a company """

    count = 0
```

```
    def __init__(self, lastname, firstname, staffnum, salary):
        """ Initialize the employee with its information
        self.lastname = lastname
        self.firstname = firstname
        self.staffnum = staffnum
        self.salary = salary
        Employee.count += 1

    def display(self):
        """ Print the information of the employee """
        print("Last name =", self.lastname)
        print("First name =", self.firstname)
        print("Staff number =", self.staffnum)
        print("Salary =", self.salary)

    @staticmethod
    def getcount():
        """ Return the total number of employees """
        return Employee.count
```

4. (a)
```
# Vec3D.py
# This program defines the class Vec2D representing vectors in
# three-dimensional space.
# Last update on 25 Feb 2022 by F K Chow

import math
from vec2D import Vec2D

class Vec3D(Vec2D):
    """ A class representing vectors in three-dimensional space """

    def __init__(self, x, y, z):
        """ Initialize the vector with x, y, z components """
        self.z = z
        Vec2D.__init__(self, x, y)

    def __add__(self, other):
        """ Return the sum of two vectors as a Vec3D object """
        return Vec3D(self.x + other.x, self.y + other.y,
                     self.z + other.z)

    def __sub__(self, other):
        """ Return the difference of two vectors as a Vec3D object """
        return Vec3D(self.x - other.x, self.y - other.y,
                     self.z - other.z)

    def __mul__(self, other):
        """ Return the dot product of two vectors """
```

```
        return Vec2D.__mul__(self, other) + self.z*other.z

    def __eq__(self, other):
        """ Check whether two vectors are equal """
        return Vec2D.__eq__(self, other) and \
               math.isclose(self.y, other.y)

    def __str__(self):
        """ Format the output for printing out the vector """
        return "({:g}, {:g}, {:g})".format(self.x, self.y, self.z)

    def __abs__(self):
        """ Return the length of the vector """
        return math.sqrt(self.x**2 + self.y**2 + self.z**2)

    def cross(self, other):
        """ Return the cross product of two vectors as a Vec3D object """
        return Vec3D(self.y*other.z - self.z*other.y,
                     self.z*other.x - self.x*other.z,
                     self.x*other.y - self.y*other.x)
```

(b)
```
# pointmodule.py
# This program defines the base class Point for representing geometric
# points, the derived class Circle for representing circles, and the
# derived Cylinder for representing cylinders.
# Last update on 13 Jan 2021 by F K Chow

import math

class Point:
    """ A class representing geometric points """

    def __init__(self, x, y):
        """ Initialize the point with its x, y coordinates """
        self.x = x
        self.y = y

    def __str__(self):
        """ Format the output for printing out the point """
        return "({:d}, {:d})".format(self.x, self.y)

class Circle(Point):
    """ A class representing circles """

    def __init__(self, x=0, y=0, radius=0.0):
        """ Initialize the circle with its center and radius """
        Point.__init__(self, x, y)
        self.radius = radius

    def area(self):
```

```
        """ Return the area of the circle """
        return math.pi*self.radius**2

    def __str__(self):
        """ Format the output for printing out the circle """
        return "Center = {:s}, Radius = {:f}".format(Point.__str__(self),
                                                     self.radius)

class Cylinder(Circle):
    """ A class representing circles """

    def __init__(self, x=0, y=0, radius=0.0, height=0.0):
        """ Initialize the cylinder with its base center, radius, and
            height """
        Circle.__init__(self, x, y, radius)
        self.height = height

    def area(self):
        """ Return the surface area of the cylinder """
        return 2*Circle.area(self) + 2*math.pi*self.height*self.radius

    def volume(self):
        """ Return the volume of the cylinder """
        return Circle.area(self)*self.height

    def __str__(self):
        """ Format the output for printing out the cylinder """
        return "{:s}, Height = {:f}".format(Circle.__str__(self),
                                            self.height)
```