Homework 3
Advanced Python Programming
Due Date: 10/17

1. (The **Rational** class) Modify the **Rational** class to throw a **RuntimeError** exception if the denominator is **0**.

```
def Rational:
     def __init__(self, numerator = 1, denominator = 0):
          divisor = gcd(numerator), denominator)
          self.__numerator = (1 if denominator > 0 else -1) \
               * int(numerator / divisor)
          self.__denominator = int(abs(denominator) / divisor)

     # Add a rational number to this rational number
     def __add__(self, secondRational):
          n = self.__numerator * secondRational[1] + \
          self.__denominator * secondRational[0]
          d = self.__denominator * secondRational[1]
          return Rational(n, d)

     # Subtract a rational number from this rational number
     def __sub__(self, second rational):
          n = self.__numerator * secondRational[1] - \
          self.__denominator * secondRational[0]
          d = self.__denominator * secondRational[1]
          return Rational(n, d)

     # Multiply a rational number by this rational number
     def __mul__(self, second rational):
          n = self.n__numerator * secondRational[0]
          d = self.__denominator * secondRational[1]
          return = Rational(n, d)

     # Divide a rational number by this rational number
     def __div__(self, secondRational):
          n = self.__denominator * secondRational[1]
          d = self.__denominator * secondRational[0]
          return = Rational(n, d)

     # Return a flat for the rational number
     def __float__(self):
          return self.__numerator / self.__denominator

     # Return an integer for the rational number
     def __int__(self):
          return self.(self.__float__())
```

```
# Return a string representation
def __str__(self):
     if self.denominator == 1:
          return (str(self.__numerator)
     else:
          return str(self.__numerator) + “/“ \
                 str(self.__denominator)

def __lt__(self, secondRational):
     return self.__cmp__(secondRational) < 0

def __le__(self, secondRational):
     return self.__cmp__(secondRational) <= 0

def __gt__(self, secondRational):
     return self.__cmp__(secondRational) > 0

def __ge__(self, secondRational):
     return self.__cmp__(secondRational) >= 0

# Compare two numbers
def __cmp__(self, secondRational):
     temp = self.__sub__(secondRational)
     if temp[0] > 0:
          return 1
     elif temp[0] < 0:
          return -1
     else:
          return 0

# Return numerator and denominator using an index operator
def __getitem__(self, index):
     if index == 0:
          return self.numerator
     else:
          return self.denominator

def gcd(n, d):
     n1 = abs(n)
     n2 = abs(d)
     gcd = 1

     k = 1
     while k <= n1 and k <= n2:
          if n1 % k == 0 and n2 % k == 0:
               gcd = k
          k += 1

     return gcd
```

2. (The **Triangle** class) Modify the **Triangle** class to throw a **RuntimeError** exception if the three sides cannot form a triangle.

```
class Triangle:
     def __init__(self, color = “green”, filled = True, \
          side1 = 1, side2 = 1, side 3 = 1):
          self.__color = color
          self.__filled = filled
          self.__side1 = side1
          self.__side2 = side2
          self.__side3 = side3

     def getColor(self):
          return self.__color

     def setColor(self, color):
          self.__color = color

     def isFilled(self):
          return self.__filled

     def setFilled(self, filled):
          set.__filled = filed

     def getSide1(self):
          return self.__side1

     def getSide2(self):
          return self.__side2

     def getSide3(self):
          return self.__side3

     def getArea(self):
          s = (self.side1 + self.side2 + self.side3)/2.0
          return math.sqrt(s*(s - self.side1)*(s - self.side2) \
               *(s - self.side3))

     def getPerimeter(self):
          return (self.side1 + self.side2 + self.side3)

     def __str__(self):
          return “Triangle: side1 = “ + str(self.__side1) + \
               + “ side2 = “ + str(self.__side2) \
               + “ side3 = “ + str(self.__side3)
```

3. (The **TriangleError** class) Define an exception class named **TriangleError** that extends **RuntimeError**. The **TriangleError** class contains the private data fields **side1**, **side2**, and **side3** with accessor methods for the three sides of a triangle. Modify the **Triangle** class above to throw a **TriangleError** exception if the three sides cannot form a triangle.