Python For Data Science

Homework Problems to Test Your Knowledge of Python

1. Download the file bobby.txt from the data files folder on canvas. Read it and count the number of distinct words. Print out an alphabetized list with the counts, i.e., the number of times “the” occurs, the number of times “cloud” occurs, etc.

2. Implement the **Euclidean Algorithm for the GCD**. Create a function **`gcd(a,b)`** that returns the greatest common divisor of **`a`** and **`b`** using the following algorithm:

   (a) **`a`** and **`b`** are integers with **`a`**> **`b`**> 0
   (b) If **`b`** is zero, print **`a`** and the algorithm is done.
   (c) if **`b`** is nonzero then there exist some number r and q such that
   $$a = bq + r$$
   where $0 \leq r < b$. Find r.
   (d) Let **`a=b`** and **`b=r`**
   (e) Print out the pair **`a,b`**
   (f) Go to step 2b.

   Implement this using a **`while`** loop (do not make a recursive call!). Your output should look something like this:

```
GCD(330, 156)
```

```
156 18
18 12
12 6
6 0
6
```

   Find: GCD(25963443,17157), GCD(6783,493305417), GCD(34880181,9877)

3. The **Sieve of Eratosthones** is an algorithm to find all the prime numbers between 1 and some integer N. It can be implemented with nested **`for`** loops:

   (a) Make a list of all the integers from 2 through N.
   (b) Cross off all the multiples of 2 (except for 2 itself). The smallest number that remains (after 2) is 3.
   (c) Cross off all the multiples of 3 (except for 3 itself). The smallest number that remains is 5.
   (d) Cross off all the multiples of 5 (except for 5 itself). The smallest number that remains is 7.
   (e) $\cdots$
   (f) Repeat looking for the smallest number p that remains and crossing off all of its multiples (except for p itself) until you reach $\sqrt{N}$.
   (g) All of the numbers that remain are prime

   Use the Sieve to print all the prime numbers under 1000.

4. Implement the geometry class exercises 30.1 - 30.4 or the vector class 30.5 - 30.7 (from *Scientific Computation: Python 3 Hacking for Math Junkies*).

Exercises

1. Define a class **point(x,y)** that represents a point in a Euclidean plane.

   (a) An **__init__** method that creates a point with attributes **x** and **y**. To invoke the method and extract the values:

   ```
   P=point(x,y)
   xvalue = P.x; yvalue = P.y
   ```

   (b) An **__repr__** method that returns the ordered pair as a string representing a list of two floats. To invoke:

   ```
   testpoint =  point(3.1,-12.5)
   print(testpoint)
   ```

   ```
   [3.1,-12.5]
   ```

2. Extend the class **point** from exercise 1 to include methods for:

   (a) Point addition and subtraction

   ```
   A=point(1,2); B=point(3,4)
   print(A+B,A-B)
   ```

   ```
   [4, 6]  [-2, -2]
   ```

   (b) Scalar multiplication from both the right and the left.

   ```
   print(3*A, B*5)
   ```

   ```
   [3.0, 6.0] [15.0, 20.0]
   ```

   (c) Testing for point equality.

   ```
   print(A==B)
   ```

   ```
   False
   ```

3. Extend the **point** class defined in exercise 1 to include methods that will do each of the following:

   (a) **Q=P.verticalReflect(y)** returns a point **Q** that is the vertical reflection of **P** across the horizontal line at **y**.

   (b) **R=P.horizontalReflect(x)** returns a point **R** that is the horizontal reflection of **P** across the vertical line at **x**.

   (c) **S=P.translate(x,y)** that returns a point **S** that translates the point **P** a distance **x** parallel to the x axis and a distance **y** parallel to the **y** axis.

   (d) **P.display(options)** to display a point in a picture using **pyplot**. By default, **display** should plot a round, 5-point diameter, red, unlabeled marker. You should allow the user to also print a label, and be able to set the label font size and offset (distance) from the point (see illustration).

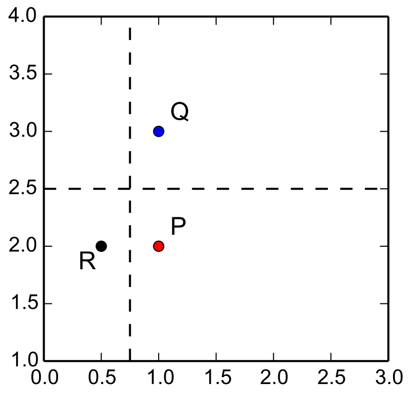
Here is an example of the code that would run the use the class created in exercise 3.

```
P=point(1,2)
P.display(label="P", \
  labeloffset=0.1)

Q=P.verticalReflect(y=2.5)
Q.display(c="blue", \
  label="Q",labeloffset=0.1)
plt.axhline(y=2.5,c="k",ls="--")

R=P.horizontalReflect(x=0.75)
R.display(c="black", \
  labeloffset=-0.2, label="R")
plt.axvline(x=.75,c="k",ls="--")

plt.xlim(0,3)
plt.ylim(1,4)
fig=plt.gcf()
fig.set_size_inches(3,3)
```



4. Define a class **triangle(A,B,C)** that takes as input three points defined by the class **point** that you defined in exercises 1 to 3. Define the methods **translate**, **horizontalReflect**, **verticalReflect** and **display** as

you did for point. The following code segment illustrates how your new class should be instantiated.

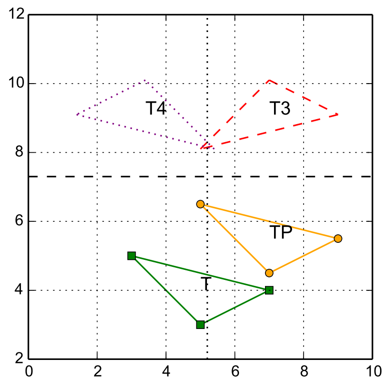
```
A=point(3,5); B=point(5,3);
C=point(7,4)

T=triangle(A,B,C)
T.display(c="green", \
  vertices=True,\
  m="s", ls="-",label="T")
TP=T.translate(2,1.5)
TP.display(c="orange",\
   vertices=True, label="TP")

T3=TP.verticalReflect(7.3)
T3.display(c="red", ls="--",\
   label="T3")
plt.axhline(y=7.3,ls="--",c="k")

T4=T3.horizontalReflect(5.2)
plt.axvline(x=5.2,c="k",ls=":")
T4.display(c="purple", ls=":",\
   label="T4")

plt.axis([0,10,2,12])
plt.grid()
```



5. A *rigid transformation* in the plane, or *isometry*, is a transformation that preserves distance. Isometries may be translations, rotations, or reflections. Every isometry in the plane can be written as a composition of mirror transformations. We can represent an isometry algebraically by representing the Euclidean plane by the $z = 1$ plane in $\mathbb{R}^3$. The point (x, y) is then represented by the column vector $[x, y, 1]^{\mathrm{T}}$. Isometries are then represented by the matrix transformation $\mathbf{Q} = \mathbf{MP}$ where either

$$\mathbf{M} = \begin{bmatrix} \cos\theta & -\sin\theta & a \\ \sin\theta & \cos\theta & b \\ 0 & 0 & 1 \end{bmatrix} \tag{30.8}$$

or

$$\mathbf{M} = \begin{bmatrix} \cos\theta & \sin\theta & a \\ \sin\theta & -\cos\theta & b \\ 0 & 0 & 1 \end{bmatrix} \tag{30.9}$$

When $a = b = 0$, (30.8) represents a rotation about the origin, and (30.9) represents a mirror reflection over a line through the origin that makes an angle $\theta/2$ with the x axis.

Use (30.9), modify the **`Triangle`** class (exercise 4) to include a method **`T.reflect(theta)`** that returns a new triangle that is reflected across a line that passes through the origin and makes an angle θ with x axis. Use the **`display`** method to verify that it works properly and illustrate the line of reflection on your plot.

6. Define a class **`vector`** that represents a three dimensional vector. Implement the following:

   (a) A **`length`** class that gives the length of vector.

   (b) A **`dotproduct`** method that returns a number.

   (c) A **`crossproduct`** method that returns a new vector.

7. Define a **`rotation`** class that represents a rotation matrix in 3 dimensions. As input, you give it a direction vector, represented by the **`vector`** class defined in exercise 1, and an angle. Then implement

   (a) A **`rotate`** method that inputs a **`rotation`** and a **`vector`** and returns a new vector.

   (b) A **`multiply`** method that inputs two **`rotation`** objects and returns a new **`rotation`** object.

   (c) A **`matrix`** method that returns the 3×3 matrix corresponding to the **`rotation`**.