# Homework 6

MPCS 51042 – Python Programming

Due: November 17th 2020, 11:59 pm

## Initial Setup

Make sure to perform a pull upstream inside your repository. This will grab the distribution code for hw6. The command is the following:

```
$ git pull upstream master
```

## Style Guide

For this homework and all future homework assignments, we will follow the style guide used by the undergraduate Python course. It's located here: https://classes.cs.uchicago.edu/archive/2020/fall/12100-1/style-guide/index.html

## Problem 1: Fraction

Place your solution to this problem in hw6/fraction/fraction.py. This file will be empty.

Write a class named `Fraction` that represents a rational number (a number that can be expressed as the quotient of two integers). It is required to implement the following methods:

- The `__init__(self, numerator, denominator)` method should accept integer values for the `numerator` and `denominator` arguments and set instance attributes of the same name. If the `denominator` is 0, raise a `ZeroDivisionError` exception. Use the `math.gcd`(https://docs.python.org/3/library/math.html#math.gcd) function to find the greatest common divisor (GCD) of the numerator and denominator and then divide each of them by it to "normalize" the fraction. For example, the fraction 28/20 will get normalized to 7/5 since the GCD of 28 and 20 is 4:

```
In [1]:x = Fraction(28, 20)

In [2]: x
Out[2]: Fraction(7, 5)
```

- Implement the basic binary operators (+, -, *, /) so that a `Fraction` can combined with either another fraction or an integer. All methods should return a new `Fraction`. Note that you may need to implement "reversed" operators for arithmetic with integers to fully work.
- The `__neg__` method should return a new 'Fraction' instance with the numerator negated.
- The `__repr__` method should return a string of the form `'Fraction(a, b)'` where a and b are the numerator and denominator, respectively.

Few test cases:

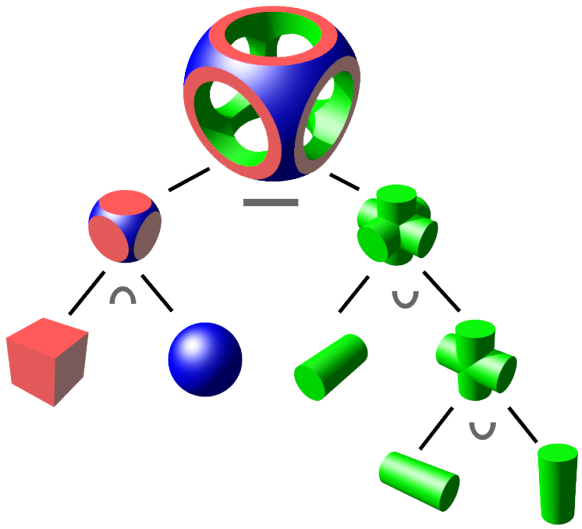
```
In [1]: from fraction import Fraction

In [2]: frac_1 = Fraction(28,20)

In [3]: frac_1
Out[3]: Fraction(7, 5)

In [4]: frac_2 = Fraction(1,0)
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
... Hiding this portion because it shows my solution
ZeroDivisionError:

In [4]: frac_2 = Fraction(1,5)

In [5]: frac_1 + frac_2
Out[5]: Fraction(8, 5)

In [6]: frac_1 * frac_2
Out[6]: Fraction(7, 25)

In [7]: frac_2 - frac_1
Out[7]: Fraction(-6, 5)

In [8]: frac_2 / frac_1
Out[8]: Fraction(1, 7)

In [9]: frac_2 / 2
Out[9]: Fraction(1, 10)

In [10]: frac_1 / 2
Out[10]: Fraction(7, 10)

In [11]: frac_1 * 2
Out[11]: Fraction(14, 5)

In [12]: -frac_2
Out[12]: Fraction(-1, 5)

In [13]: 2 - frac_2
Out[13]: Fraction(9, 5)

In [14]: 2 - (-frac_2)
Out[14]: Fraction(11, 5)
```

## Problem 2: CSG

Place your solution to this problem in hw6/csg/csg.py.

In this problem, you will need to exercise your knowledge of operator overloading in Python to build an application that produces images of complex geometric objects via a technique called constructive solid geom-

etry(CSG). More information can be found at this link: (https://en.wikipedia.org/wiki/Constructive_solid_geometry). CSG allows one to model arbitrary geometric objects by representing them as Boolean operators applied to simple *primitives* (basic geometric shapes). Namely, objects are represented as binary trees where the leaves are primitive shapes (spheres, cylinders, etc.) and the nodes are operators (intersection, union, and difference).



You are asked to build an application that draws two-dimensional CSG objects. You will need to build two classes representing primitives (`Circle` and `Rectangle`) and three classes representing operators (`Intersection`, `Union`, and `Difference`). You will also need to define an abstract base class (`Drawable`) that the primitives subclass, which provides both an interface (a set of abstract methods that the subclasses must implement) and abstract methods, such as a `draw()` method that draws the shape.

To visualize the complex shapes that are represented by CSG binary trees, we will use the built-in tkinter(https://docs.python.org/3/library/tkinter.html) standard library module, which provides Python bindings to Tcl/Tk. The basic logic for creating a window and a canvas on which the image will be drawn has already been set up for you. In addition, a `draw_pixel(...)` function has been provided that draws a single pixel at a given (x,y) location. Note that on the Tk canvas, the top-left corner of the window corresponds to the (0,0) location.

The classes representing primitives and operators are all required to have a `__contains__()` method that allows one to use the `in` operator to check whether a given (x,y) point is inside the corresponding shape:

```
In [1]: circ = Circle(x=0., y=0., r=5.)

In [2]:  (1, 1) in circ
Out [2]: True

In [3]: (10, -5) in circ
Out [3]: False
```

A simple method to draw an image of a shape then is to iterate over all pixels in a grid, check whether each point is `in` the shape, and if it is color the pixel.

## Specifications

Detailed specifications for each class are listed below. In addition, each class (other than `Drawable`) must have a `__repr__()` method that gives a string representation of the class. You can determine the string representation. There is no required format.

## Drawable Class

- `Drawable` should be an abstract base class.
- It should have a `__contains__` method that is an `@abstractmethod`.
- The `__and__(self, other)` method (which overloads the `&` operator) should return an instance of `Intersection`, representing the intersection of the two operands.
- The `__or__(self, other)` method (which overloads the `|` operator) should return an instance of `Union`, representing the union of the two operands.
- The `__sub__(self, other)` method should return an instance of `Difference`, representing the difference between the two operands.
- The `draw(self, canvas)` method accepts an instance of `tkinter.Canvas` and draws the shape represented by `self`. The physical coordinate (0,0) should appear in the center of the canvas. Thus, you must translate from the canvas coordinates to the physical coordinates of the CSG objects. As an example, if the canvas is 100 by 100, then the canvas location (50,50) would correspond to a location of (0,0) in physical coordinates, since (50,50) represents the center of the canvas. Similarly, the canvas location (0,0) would correspond to a physical location of (-50,50).

To draw a shape, loop over each (x,y) pixel in the canvas and check whether the corresponding physical coordinate is `in` the shape, i.e., check whether `(x,y) in self` and if so, call the `draw_pixel()` function that has been provided to you.

Note: to get the width and height of the canvas, you can index it as `canvas['height']` and `canvas['width']` (both return strings, not integers).

## Circle Class

- The `Circle` represents a circle of a given radius centered at a given location.
- The `__init__(self, x, y, r)` method takes the x- and y-coordinate of the center of the circle and its radius and stores them in instance attributes.
- The `__contains__(self, point)` method takes a list or tuple of two items, representing an (x,y) coordinate, and determines whether that point is inside the circle.

## Rectangle Class

- `__init__(self, x0, y0, x1, y1)` method accepts the lower-left (x0,y0) and upper-right (x1,y1) coordinates of the rectangle and stores them in instance attributes.
- `__contains__(self, point)` method takes a list or tuple of two items, representing an (x,y) coordinate, and determines whether that point is inside the rectangle.

## Operator Classes

- Define three operator classes: `Intersection`, `Union`, and `Difference`.
- For each operator class, the `__init__(self, shape1, shape2)` accepts two arguments corresponding to two shapes and stores them in instance attributes. Each should also define a `__contains__` method.
- The`__contains__(self, point)` method of the `Intersection` class returns `True` if the `point` is in both shapes specified in the initializer.
- The `__contains__(self, point)` method of the `Union` class returns `True` if the `point` is in at least one of the shapes specified in the initializer.

- The `__contains__(self, point)` method of the `Difference` returns `True` if the `point` is in the first shape but not in the second.
- Don't overthink these implementations they are relatively similar and simple.

## Inheritance and Testing

Part of your grade for this assignment is for you determine what the classes should inherit another class. I won't help with questions specifically about this portion of the assignment. You need to determine whether a class should or should not inherit. However, the main method that uses these classes should help with figuring this portion out.

I want you to test your own classes. Most of them are relatively straight forward but if you have any questions then please let me know via Piazza.

## main() function

The `main(shape)` function accepts an instance of `Drawable`, creates a window/canvas and should draw the shape by calling its `draw()` method. Most of this function has already been written for you (namely, creating the window/canvas and calling the event loop).

## Lastly...

One example shape that will use your classes has been been set up already in the `__name__ == '__main__'` section. Once you're done building all the classes, draw a "happy face" by subtracting two eyes and a mouth from a head.