Problem 1
Write a class Car that has the following attributes:
- Model
- Year
- Color
Make sure that the attributes are private within the class.

Write the method that initializes the object.
Write getter methods and setter methods for each of the attributes.
Write a method called compare_cars(self, car2) that will take another car object as a parameter. The method will return True if both cars have the same Model and Year. It will return False otherwise.

class Car:
    def __init__(self, model, year, color):
        self._model = model
        self._year = year
        self._color = color

    def get_model(self):
        return self._model
    def get_year(self):
        return self._year
    def get_color(self):
        return self._color

    def set_model(self, new_model):
        self._model = new_model
    def set_year(self, new_year):
        self._year = new_year
    def set_color(self, new_color):
        self._color = new_color
def compare_cars(self, other):
    return self._model == other.model and self._year == other._year
Problem 2
An object consists of the following:
- A collection of values
- A set of behaviors

How are an object's collection of values represented in a Python class?

By the various fields inside it. (Each field is a variable stored inside the object.)

How are an object's set of behaviors represented in a Python class?

By the various methods of the class. (These are functions “inside” an object- they take ‘self’ parameters, and do things to the object when called.)
Problem 3
Write a class Polygon that has the following attributes:
- A list of tuples, denoting the x,y coordinates for each of the points of the shape.
- A “line color” (the color to use to draw the outer edge). You don’t need to know the type of this parameter.
- A “fill color” (the color used to draw the inside edge).

(Make sure to include all of the necessary getters.)

All Polygons are closed shapes - that is, the last line should connect back to the first point.
Should you store that first point once or twice (there are good arguments for both)?
- Discuss with your group what design to use; choose one or the other.
- In the docstring for __init__(), document what format users should use in the parameters
  - It doesn’t have to be the same as your internal design!
- Document your internal design in your class, in the comments

Assume that I’ve provided you a function draw_line(x1,y1, x2,y2, color) which draws individual lines. Write a draw_edge() method which draws the outer edge of the polygon.

Suppose that you want to write a function which will determine whether two Polygons overlap. This is not an easy function to write, so I won’t assign it to you - but provide the arguments for the function, and write its docstring. What should its parameters be, and what should it return? (There are a variety of plausible answers.)

Write a function (not a method of the Polygon class) called square() which makes it easy to build squares out of Polygon objects. What should its parameters be? Write the code which creates and returns a new Polygon.

class Polygon:
    # the points in this polygon do *NOT* duplicate the end-point.
    # So, the various edges of the polygon are 0-1, 1-2, 2-3 (etc.)
    # and end with n-0.

    def __init__(self, pts, line_color, fill_color):
        """Initializes the Polygon object.

        Parameters: points, line_color, fill_color
        The points should be a list of (x,y) tuples. The list should
        *NOT* duplicate the first/last point. For example, a triangle
        should be defined as a list with three elements.

        Pre-conditions: There must be at least three points"

        # this assert is a good idea, but you hadn't learned about
        # asserts at this point. So your solution didn't have to
        # have it.
        assert len(pts) >= 3
self._pts = pts
self._line_color = line_color
self._fill_color = fill_color

def draw_edge(self):
    # draw the n-0 line first. Conceptually, I think of it as the
    # last line, but order doesn't matter (for this algorithm), and
    # it's easier to do the *special* case before the loop.
    a = pts[-1]
b = pts[0]
draw_line(a[0],a[1], b[0],b[1], line_color)

    for i in range(len(pts)-1):
        a = pts[i]
b = pts[i+1]
draw_line(a[0],a[1], b[0],b[1], line_color)

def overlaps(self, other):
    """Checks to see if two Polygon objects overlap. This will
    treat 'touching at a vertex' or 'touching at an edge' as *NOT*
    overlapping; there must be non-zero area in common. That is,
    the polygons overlap if their intersection is one or more
    non-point polygons.

    Parameters: other polygon

    Return value: True or False"
    TODO("implement me")

def square(x,y, radius, line_color, fill_color):
    """Creates a Polygon object which is an axis-aligned square.

    Parameters: x,y, radius, line_color, fill_color
    The x,y values are the center of the square (given as two numbers,
    *NOT* a tuple)
    The radius is the distance from the center of the square to one edge.""

    lower_left  = (x-radius, y-radius)
    lower_right = (x+radius, y-radius)
    upper_left  = (x-radius, y+radius)
    upper_right = (x+radius, y+radius)
    return Polygon([lower_left, upper_left, upper_right, lower_right],
                   line_color, fill_color)
Problem 4

You're given the following problem statement:
“You are to design a reverse address lookup system that reads in a file of people’s addresses and
organizes this data so that, given an address, we can figure out who lives there.”

Input format
The input is in a CSV file. Each line in this file is an individual’s address, and has the following
format, with individual fields separated by commas:

Name,Street Address,City,State

e.g.:
John M. Brown,1212 Main Street,Tucson,AZ
Maria De Costa,3472 E Calle Concordia,Phoenix,AZ
Felix Gutierrez,932 NW Main Street,Port Jefferson,NY
Mary Brown,1212 Main Street,Tucson,AZ

We need to decompose the problem and identify the structure of the code, think about what kind
of data you need to store, and how you're going to store it. Discuss with the people around you on
how to solve this problem. Answer the following:
- What are the top-level tasks in this problem?
- For each top level task, decompose them to the next level of tasks
- Repeat this process for another level

(answers will vary here. The answers below are an example - not the only possibility.)

The top-level tasks (and their respective next-level decompositions) might be:

- Open the file
  - Get file name from user
  - Attempt to open the file
  - Check for errors
- Read the various lines of the file, store them into a data structure
  - Read each line in the file
  - Split line into fields
  - Check for incorrect data format
  - Store into data structure
- Lookup and report the results
  - Search through data for a match on the address
  - Print out report

Instructor’s Note:
You will notice that several of the points up above are quite vague, because we don’t yet know how the
data is organized - or maybe, because the algorithm is not trivial. That’s OK! That’s kind of the point.
Those vague, complex parts are exactly where we would work on doing more decomposition.
When we read the data from a file into memory, there are several ways to organize it. Give at least three different ways we might store the data in memory. For each one, explain it in enough detail that another Python programmer could create (or use) the data structure without confusion.

- A list of tuples: each tuple represents one record in the file, with the fields of the tuple representing the comma-separated fields from the data file.
  - (We probably would do a little bit of input validation, and make sure that the data looks valid - so that addresses would always be at the same element of the tuple. Or we might just assume that the CSV is already well-designed.)
- A dictionary, where the keys are the names of people, and the other fields (as a tuple) are the values.
- A dictionary, where the keys are the **addresses**, and the values are tuples containing the other fields (including the name)
  - Should the address contain only the street address, or the entire address - street, city, state? That’s a question to delve into further...

**Different ways are optimized for different programs.**

- Assume that the only thing you will do is reverse-address-lookup. What sort of organization makes that the fastest?
- Assume that (sometimes) you will be sorting based on last name. What sort of organization might make that easy to do, and fast?

- For reverse-address lookup, the third option above is best, as we can simply search for “address in dictionary.” We’d have to decide whether the users would be looking for street addresses, or have the full 3-part address. Also, if we needed to support search for imprecise addresses, we’d need a more complex design.
- If we were sorting based on last name, probably the list of tuples would be best - but we’d separate each name into (last, first) tuples (and put that tuple first). If we did this, then a simple .sort() method on the list would sort all of our records.

**Finally, discuss your assumptions about the data.** For instance, did you assume that there were no duplicate records? Did you assume that only one person was at each address, or could there be multiple people? How would you change your data structure if you made different assumptions?

I generally assumed that there were no duplicate records, and also that there was only one person at an address. The list-of-tuples allows maximum flexibility for this sort of situation - duplicates are handled easily - but the dictionaries don’t work well if you might have multiple records with the same key.

To handle duplicates (or multiple-people-in-a-house), we’d probably upgrade the dictionary implementations to store **lists** of tuples instead of simple tuples. We’d only add a key to the dictionary when we found a record which needed it - but when we did so, we would add a **single element list** as the value. Then, if we found other records with the same key, would could just append to the list.