Assignment 11

Objectives: Play with snippets of code using Object Oriented Programming. Write classes and methods using inheritance. Learn about the super() built-in function.

Note: Include DocStrings in each script you submit analogous to the following:

```
"""Defines some classes for a future apartment search app


Submitted by Mauricio Arias. NetID: ma6918
A main Apartment class is defined and some subclasses defined.
There are two main classes of apartments defined at this point.
Studio: a single room which sometimes has a pseudo-division for the kitchen.
Two-bedroom apartment: an apartment with two bedrooms, which sometimes includes a
roommate, leaving only one room to be rented out.
"""
```

Make single-line DocStrings for all methods and multiline DocStrings for all classes. For the latter use a similar format as for the DocStrings used for the scripts: in the lower part of the DocString include a simple description for each attribute and a simple description for each method: in many cases PyCharm should prompt you.

Background

The built-in function super() has been the subject of a lot of controversy. It enables access to methods from parent classes but some decisions about implementation have caused issues over the years. In the case of single inheritance, where only one parent class is defined, it works well. However, its implementation relies on Python figuring out what the parent class is by the place and manner in which it is called. This is fairly simple with single inheritance. However, if multiple parent classes have been defined, Python has to figure out which one should be chosen for this particular call. Some decisions have been made in this respect but many aspects of Python philosophy, including simplicity and error-avoidance, have been put at risk. (Some languages avoid multiple inheritance altogether and provide alternative tools to mimic its functionality.) Here is an example of its use for single inheritance, which is what we are doing for this assignment.

```
class ElectricCar(Car):
    def __init__(self, color, price, battery_size = 100, voltage = 300):
       super().__init__(color, price)
       self.battery_size = battery_size
       self.voltage = voltage
```

Part 1. **Inheritance**
Make three classes: Apartment, Studio, and TwoBedApartment. The last two classes inherit from the first one. They should all be in a single module. (You can think of this as the beginning of a project to allow users to find apartments according to budgets and specific needs. We are starting with only a couple types of apartments.)

When importing into the main script, import only the child classes.

Apartment (3 points)
Methods:

1. The constructor takes two arguments: ID (an assigned string external to this script) and rent (monthly rent in dollars)
2. The constructor specifies the following attributes:
   a) ID
   b) monthly rent
   c) the area in sq. ft. (assign 0 at construction; this is the total area of the apartment)
3. A setter method for the area. It takes a single number corresponding to the area in square feet.
4. A getter method to report the area.
5. A method to report the monthly price per unit of area. If the area is 0, it should report -1 to indicate that there is an error. (Raising an error would be a better option but we will only report the situation for simplicity.)
6. A general method for the target wages according to the following rule of thumb:
   monthly rent * 12 / 0.4

Studio (2 points)
Methods:

1. The constructor takes three arguments: ID (a number assigned somewhere else), rent (monthly rent in dollars) and whether it has a separate kitchen (as opposed to a kitchenette; defaults to yes).
2. The constructor specifies the following attributes (use the built-in function super() to extend the original constructor):
   a) ID
   b) monthly rent
   c) the area in sq. ft.
   d) separate kitchen with a boolean value (True indicates a separate kitchen and False indicates a kitchenette.)
3. A setter method to mark it as having a separate kitchen space.
4. A getter method to report whether it has a separate kitchen.

Two-Bedroom Apartment (3 points)

Methods:

1. The constructor takes three arguments: ID (a number assigned somewhere else), rent (monthly rent in dollars) and whether it comes with a roommate (boolean, default to yes).
2. The constructor specifies the following attributes (use the built-in function super() to extend the original constructor):
   a) ID
   b) monthly rent
   c) the area in sq. ft.
   d) a dictionary with the areas for the “common areas,” “room 1” and “room 2.” (Initial values are 0.)
   e) whether it has someone living there already: a roommate. The roommate is assumed to be in the **first** room
3. A setter method to populate the dictionary with the areas of the different rooms. For simplicity it takes a list of areas and assigns the first two numbers to the first room and second room, respectively. It calculates the common area by using the total area. If the last one comes out as less than 20% of the areas of the rooms combined, it reports an error and quits. (Raising an InsufficientArea exception will be better here. Do it if you want to.)
4. A method to set the area that overrides the inherited one. If the areas for the different parts of the apartment have been set, the common area will have to be modified to accommodate the new total area. It should use the method above so we don’t duplicate how to handle area issues: design your strategy and implement it.
5. A setter method to indicate whether it comes with a roommate.
6. A getter method to report whether it comes with a roommate.
7. A setter method to add the roommate information: family name and given name. (Note: If there is no roommate before this assignment, it changes that attribute to True.)
8. A method to report the monthly price per unit of area. If the area is 0, it should report -1 to indicate there is an error. (Raising an exception would be a better option but we will do this for simplicity. Raise an exception if you want to.) If the apartment includes a roommate, the area to be used is half the area of the common areas and the area of the second room (assumed for simplicity). This method overrides the inherited one.

Main script (3 points)

In a separate file, write a script that makes a list of apartments according to user input. The number of apartments shoudl not be hardcoded in the script: a way to handle this should be incorporated in the main script. Request the type of apartment and depending on the type create an object and fill-in the appropriate information.
Test it with four apartments according to the following specifications:

Apt 1 (ID: xr1234):

Studio with kitchen for $1600 with an area of 400 sq. ft.

Apt 2 (ID: as4545):

Studio with kitchenette for $1300 with an area of 300 sq. ft.

Apt 3 (ID: hj455):

A 2-bedroom apt with areas 500 sq. ft (common), 150 sq. ft (room 1) and 140 sq. ft (room 2). The rent is $2200.

Apt 4 (ID: hj456):

A 2-bedroom apt with areas 400 sq. ft, 130 sq ft. and 160 sq. ft. The rent is $1100 for the unoccupied room. The roommate is Balki Bartokomous (given-name family-name).

Print the information specified below about each apartment by looping over the list of apartments.

- apartment ID
- type of apartment: the implementation of this is up to you. Make it simple and intuitive to use. Add it to your classes.
- price
- total area
- price per area (notice that the area here is not the total area if there are roommates)
- target wages/income

If, and only if, there are apartments with roommates, include a note warning that the price per area uses an effective area to take into account that some areas are shared.

Name the module `[NetID]_apartments.py` and the main script `[NetID]_apt_search.py`.