Implementing a class

Step 1.

In order to implement a class, the first step is to list the responsibilities of the objects. In this case, our **Country** class should have these responsibilities:

1. Display the menu.
2. Get user input (name, population and area)
3. Store the list of countries and its attributes.
4. Calculate the country with largest area.
5. Calculate the country with the largest population.
6. Calculate the country with the largest population density.

Step 2

The second step, is to specify the public interface of the **Country**, in these case there are 6 methods to be implemented in the public interface:

```
          Usage
--------------------------------
country1 = Country('a',10,10) # initialize a Message
country1.largestCountryDict()
country1.largestCountryList()
country1.mostPopulatedCountryDict()
country1.mostPopulatedCountryList()
country1.largestDensityCountryDict()
country1.largestDensityCountryList()
```

Step 3

Every method and class is documented as follows:

```
# This module defines the Country class. A country has a name, a population and an area
25 usages   S1089563
class Country:
    _dictCountries = dict()
    _listCountries = []
    ## Constructor method for the class Country. it receives the contries name
    ## @param name: name of the country
    ## @param population: population of the country
    ## @param area: area of the country
    S1089563
    def __init__(self, name, population, area):
```

And for each of the 6 methods, the comment goes as follows:

```
## Returns the country with largest population
## @return country with the largest population
1 usage   S1089563
@classmethod
def mostPopulatedCountryDict(cls):
```

Step 4

In this case, since the **Country** has a name, an area and a population, these 3 attributes are created as instance variables:

a. _name
b. _population
c. _area

In addition to the instance variables, two additional class variables are created to store the information of each country, one as a dictionary and the other one as a list:

a. _dictCountries

b. _listCountries

Step 5

In the constructor of the class, the instances and class variables are initialized:

```
def __init__(self, name, population, area):
    self._name = name
    self._population = population
    self._area = area
    Country._dictCountries[name] = {'area': area, 'population': population}
    Country._listCountries.append([name, area, population])
```

Step 6

After the constructor is implemented, each of the 6 methods are created. In this case, the implementation is the same for each method. Since we are looking for a maximum, each method will have two auxiliary variables to store the local max and the name of the country.

```
## Returns the country with largest population
## @return country with the largest population
1 usage   S1089563
@classmethod
def mostPopulatedCountryDict(cls):
    maxPopulation = -1000
    largestCountry = ''
    for country in Country._dictCountries:
        if Country._dictCountries[country]['population'] > maxPopulation:
            maxPopulation = Country._dictCountries[country]['population']
            largestCountry = country
    return print(largestCountry)
```

Depending on the type (dictionaries or lists) slight differences are made:

```
## Returns the country with the largest population
## @return conutry with the ñargest population
1 usage   S1089563
@classmethod
def mostPopulatedCountryList(cls):
    largestCountry = ''
    maxPopulation = -1000
    for country in Country._listCountries:
        if country[2] > maxPopulation:
            largestCountry = country[0]
            maxPopulation = country[2]
    return print(largestCountry)
```

Step 7

Finally, in order to test the class, a different python script is used to test the class in isolation. To do this correctly, the argparse method is used and a demo is implemented:

```
           Country
-------------------------------
A simulated Country program. Stores countries and returns the largest country,
the most populated country and the most dense country:
1) largestCountryDict:  returns the largest country (using dictionaries)
   @return largest country
5) largestDensityCountryDict(): returns the largest density country (using dicts)
   @return largest density country

6) largestDensityCountryList(): returns the largest density country (using lists)
   @return largest density country

optional arguments:
  -h, --help  show this help message and exit
  --run_demo  runs this demo

          Usage
-------------------------------
 country1 = Country('a',10,10) # initialize a Message
 country1.largestCountryDict()
 country1.largestCountryList()
 country1.mostPopulatedCountryDict()
 country1.mostPopulatedCountryList()
 country1.largestDensityCountryDict()
```

After executing the demo, the results show that all the test performed have passed:

```
if args.run_demo:
    country1 = Country( name: 'a',  population: 1000,  area: 1)
    country2 = Country( name: 'b',  population: 1,  area: 1000)
    Country.largestCountryDict()
    print('Expected: b')
    Country.largestCountryList()
    print('Expected: b')
    Country.mostPopulatedCountryDict()
    print('Expected: a')
    Country.mostPopulatedCountryList()
    print('Expected: a')
    Country.largestDensityCountryDict()
    print('Expected: a')
    Country.largestDensityCountryList()
    print('Expected: a')
```

```
b
Expected: b
b
Expected: b
a
Expected: a
a
Expected: a
a
Expected: a
a
Expected: a
```