BME 646 / ECE 695DL: Homework 1

Yuan-Yao Lou
10 January 2022

1 Introduction

This homework aims to practice the concept of Object-Oriented (OO) in Python. Besides, future homework assignments would be specific using PyTorch, which intensively uses Python OO.

2 Methodology

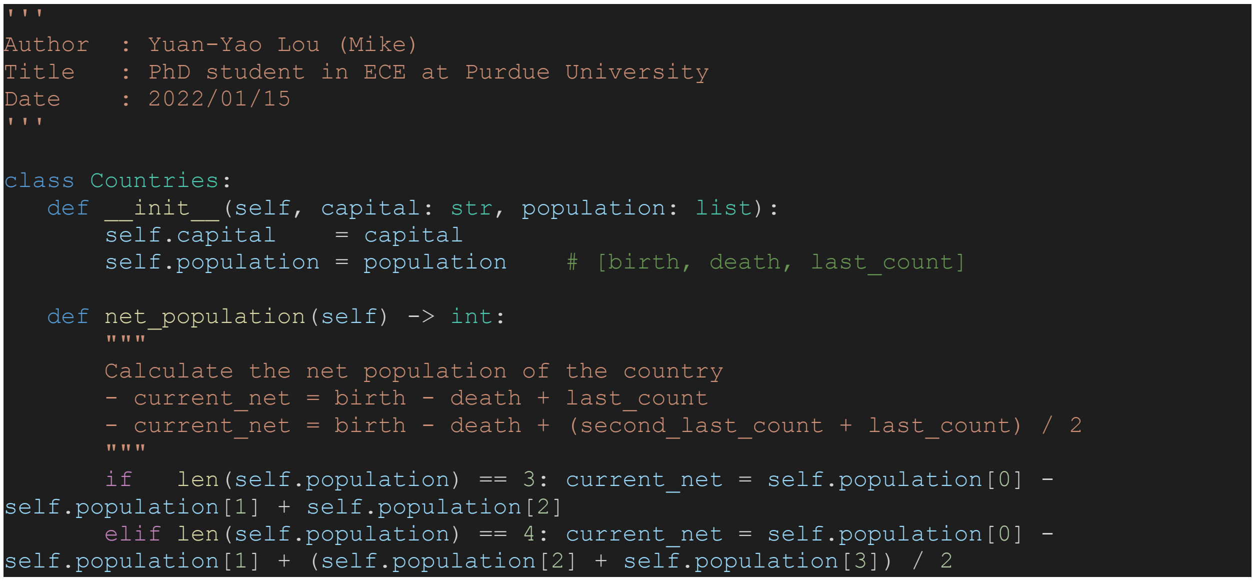
In this homework, we adopt two Python OO mechanisms, **inheritance and function overwriting,** in two classes, which are the parent class "Countries" and the child class "GeoCountry".

The class "GeoCountry" overwrites the function "net_population", which is already defined in the parent class "Countries", to overcome the issue that leads to a margin of error that skews the data.

Other than that, **the child class "GeoCountry" uses "super()" in three class methods** to get part of the work done by class methods defined by its superclass "Countries". One of the class methods that uses "super()" is "__init__()" for the class initialization, and the other twos are "density_calculator1()" and "density_calculator2()" for the calculation of density.

Last but not least, the function "net_density()" in the class "GeoCountry" **returns a callable object** related to the density calculation instead of the outcome (i.e., density).

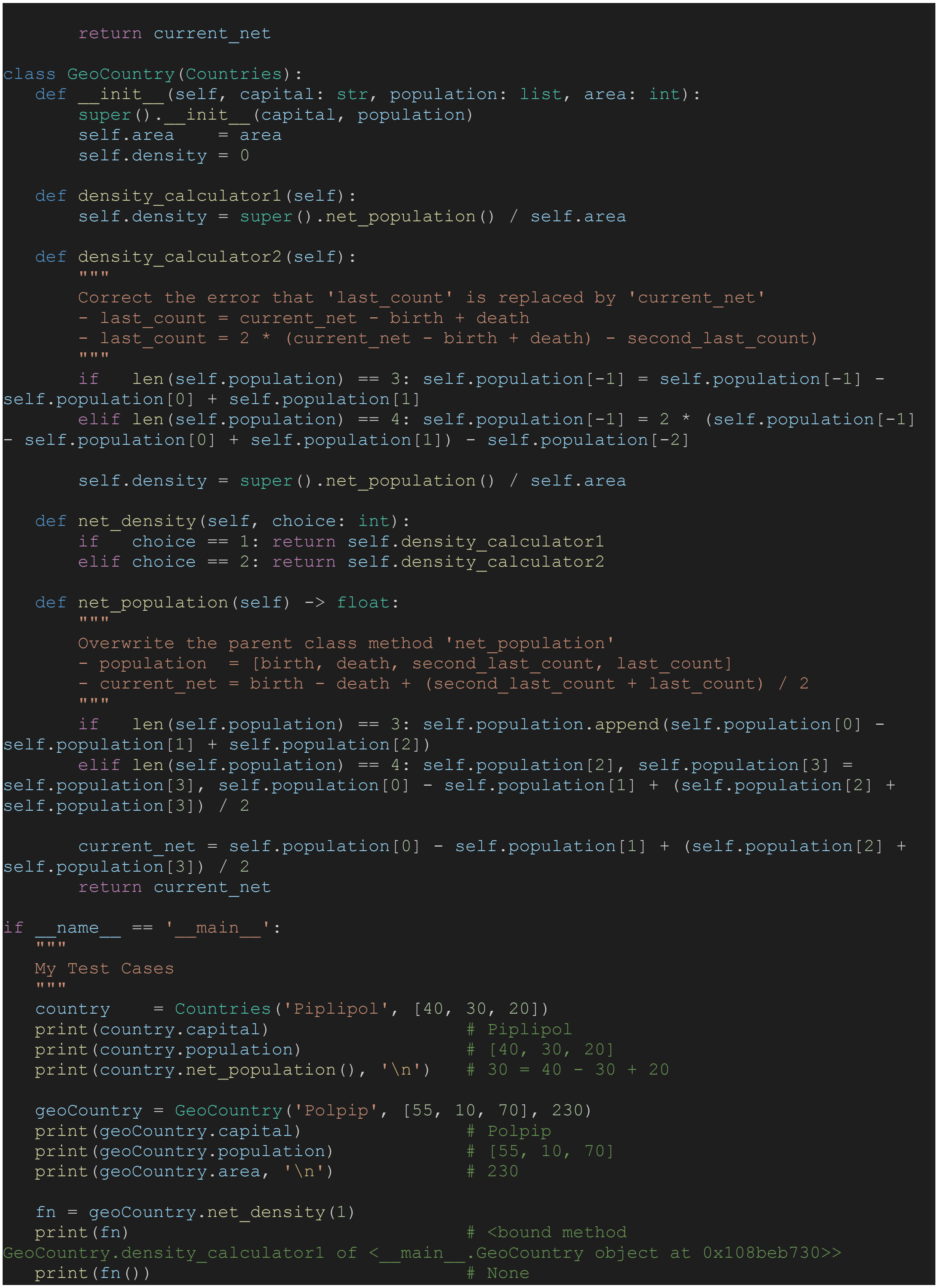
3 Implementation and Results



```
'''
Author   : Yuan-Yao Lou (Mike)
Title    : PhD student in ECE at Purdue University
Date     : 2022/01/15
'''

class Countries:
   def __init__(self, capital: str, population: list):
       self.capital    = capital
       self.population = population    # [birth, death, last_count]

   def net_population(self) -> int:
       """
       Calculate the net population of the country
       - current_net = birth - death + last_count
       - current_net = birth - death + (second_last_count + last_count) / 2
       """
       if   len(self.population) == 3: current_net = self.population[0] -
self.population[1] + self.population[2]
       elif len(self.population) == 4: current_net = self.population[0] -
self.population[1] + (self.population[2] + self.population[3]) / 2
```



```
       return current_net

class GeoCountry(Countries):
   def __init__(self, capital: str, population: list, area: int):
       super().__init__(capital, population)
       self.area    = area
       self.density = 0

   def density_calculator1(self):
       self.density = super().net_population() / self.area

   def density_calculator2(self):
       """
       Correct the error that 'last_count' is replaced by 'current_net'
       - last_count = current_net - birth + death
       - last_count = 2 * (current_net - birth + death) - second_last_count)
       """
       if   len(self.population) == 3: self.population[-1] = self.population[-1] -
self.population[0] + self.population[1]
       elif len(self.population) == 4: self.population[-1] = 2 * (self.population[-1]
- self.population[0] + self.population[1]) - self.population[-2]

       self.density = super().net_population() / self.area

   def net_density(self, choice: int):
       if   choice == 1: return self.density_calculator1
       elif choice == 2: return self.density_calculator2

   def net_population(self) -> float:
       """
       Overwrite the parent class method 'net_population'
       - population  = [birth, death, second_last_count, last_count]
       - current_net = birth - death + (second_last_count + last_count) / 2
       """
       if   len(self.population) == 3: self.population.append(self.population[0] -
self.population[1] + self.population[2])
       elif len(self.population) == 4: self.population[2], self.population[3] =
self.population[3], self.population[0] - self.population[1] + (self.population[2] +
self.population[3]) / 2

       current_net = self.population[0] - self.population[1] + (self.population[2] +
self.population[3]) / 2
       return current_net

if __name__ == '__main__':
   """
   My Test Cases
   """
   country    = Countries('Piplipol', [40, 30, 20])
   print(country.capital)                  # Piplipol
   print(country.population)               # [40, 30, 20]
   print(country.net_population(), '\n')   # 30 = 40 - 30 + 20

   geoCountry = GeoCountry('Polpip', [55, 10, 70], 230)
   print(geoCountry.capital)               # Polpip
   print(geoCountry.population)            # [55, 10, 70]
   print(geoCountry.area, '\n')            # 230

   fn = geoCountry.net_density(1)
   print(fn)                               # <bound method
GeoCountry.density_calculator1 of <__main__.GeoCountry object at 0x108beb730>>
   print(fn())                             # None
```

```
    print(geoCountry.density)                # 0.5 = (55 - 10 + 70) / 230

    geoCountry.population = [55, 10, 115]
    fn = geoCountry.net_density(2)
    print(fn)                                # <bound method
GeoCountry.density_calculator2 of <__main__.GeoCountry object at 0x108beb730>>
    print(fn())                              # None
    print(geoCountry.density, '\n')          # 0.5 = (55 - 10 + (115 - 55 + 10)) / 230

    geoCountry = GeoCountry('Polpip', [55, 10, 25], 230)
    print(geoCountry.net_population())       # 92.5
    print(geoCountry.population)             # [55, 10, 25, 70]
    print(geoCountry.net_population())       # 126.25
    print(geoCountry.population)             # [55, 10, 70, 92.5]
    print(geoCountry.net_population())       # 154.375
    print(geoCountry.population, '\n')       # [55, 10, 92.5, 126.25]

    """
    TA's Test Cases
    """
    ob1 = GeoCountry('YYY', [20,100, 1000], 5)
    print(ob1.density)                       # 0
    print(ob1.population)                    # [20, 100, 1000]
    ob1.density_calculator1()
    print(ob1.density)                       # 184.0
    ob1.density_calculator2()
    print(ob1.population)                    # [20, 100, 1080]
    print(ob1.density)                       # 200.0

    ob2 = GeoCountry('ZZZ', [20, 50, 100], 12)
    fun = ob2.net_density(2)
    print(ob2.density)                       # 0
    fun()
    print("{:.2f}".format(ob2.density))      # 8.33

    print(ob1.population)                    # [20, 100, 1080]
    print(ob1.net_population())              # 960.0
    print(ob1.population)                    # [20, 100, 1080, 1000]
    print(ob1.density)                       # 200.0

    ob1.density_calculator1()
    print(ob1.population)                    # [20, 100, 1080, 1000]
    print(ob1.density)                       # 192.0
    ob1.density_calculator2()
    print(ob1.population)                    # [20, 100, 1080, 1080]
    print(ob1.density)                       # 200
```

4 Lessons Learned

Practicing Python OO through this homework helps me learn **how to create a class hierarchy via inheritance** and **how to clearly define their relationship.** Moreover, **overwriting the function** defined in the parent class via the child class gives me insights into the high flexibility of Python programming. I also learn **how to return a callable object in a method.**

5 Suggested Enhancements

For this homework (Python OO practice), I think there is no further enhancement required.