BME 646/ ECE695DL: Homework 1

Varun Aggarwal

January 18, 2022

1 Introduction

This project was aimed at brushing over the basics of Objects and Classes in Python (the building blocks of Object Oriented Programming). In addition, a vital concept of Inheritance was covered. Overall, this HW had seven tasks.

2 Methodology

The seven tasks in this assignment are tackled as follows:

Task1: Class *Countries* is created with two instance variables.

Task2: Created an object of class *Countires*

Task3: A new function is added to the class *net_population* which returns current net population based on the formula $birth - death + last\_count$

Task4: Created a subclass *Geocountry*

Task5: Created an object of class *Geocountry*

Task6: Three new functions are created. These functions were used to calculate the population density. While it was straightforward to implement them in the beginning, they were later modified in Task 7.

Task7: function *new_population* was created with the same name as the function in parent class of *Geocountry.* Function calls were resolved by explicitly calling parent class's function by the use of keyword *Countries* and inherited class's function by using keyword self. In addition, density calculation functions were modified to to handle population with length 3 and 4.

3 Implementation and Results

CODE

```
class Countries:
    def __init__(self, capital, population):
        self.capital = capital
        self.population = population

    def net_population(self):
        # birth - death + last_count
        current_net = self.population[0] - self.population[1] + self.
            ↪ population[2]
        return current_net


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

    def density_calculator1(self):
        if len(self.population) == 3:
            # calling parent function
            self.density = Countries.net_population(self) / self.area
        else:
            self.density = self.net_population() / self.area

    def density_calculator2(self):
        """
        update population, call density_calculator1
        new_last_count = last_count +death - birth
        """
        if len(self.population) == 3:
            self.population[2] = (
                self.population[2] + self.population[1] - self.population[0]
            )
        else:
            self.population[3] = (
                self.population[3] + self.population[1] - self.population[0]
            )
        self.density_calculator1()

    def net_density(self, choice):
        if choice == 1:
            return self.density_calculator1
        elif choice == 2:
            return self.density_calculator2
        else:
            print("Incorrect␣Choice")
            return

    # TASK 7: add net_population
    # let it handle population length = 4
    def net_population(self):
        # only do this when length is 3
        if len(self.population) == 3:
            self.population.append(Countries.net_population(self))
        current_net = (
            self.population[0]
            - self.population[1]
            + (self.population[2] + self.population[3]) / 2
        )
        return current_net


if __name__ == "__main__":
    # initialized, never used
    obj1 = Countries("Piplipol", [40, 30, 20])
    obj2 = GeoCountry("Polpip", [55, 10, 70], 230)
```

TEST CASE

```
ob1 = GeoCountry("YYY", [20, 100, 1000], 5)
print(ob1.density)
print(ob1.population)
fn = ob1.net_density(1)
fn()
print(ob1.density)
fn = ob1.net_density(2)
fn()
print(ob1.population)
print(ob1.density)
```

```
ob2 = GeoCountry("ZZZ", [20, 50, 100], 12)
fun = ob2.net_density(2)
print(ob2.density)
fun()
print("{:.2f}".format(ob2.density))
print(ob1.population)
print(ob1.net_population())
print(ob1.population)
print(ob1.density)
ob1.density_calculator1()
print(ob1.population)
print(ob1.density)
ob1.density_calculator2()
print(ob1.population)
print(ob1.density)
```

OUTPUT

```
>>> 0
>>> [20,100,1000]
>>> 184.0
>>> [20,100,1080]
>>> 200.0
>>> 0
>>> 8.3
>>> [20,100,1080]
>>> 960.0
>>> [20,100,1080, 1000]
>>> 200.0
>>> [20,100,1080, 1000]
>>> 192.0
>>> [20,100,1080, 1080]
>>> 200.0
```

4 Lessons Learned

This programming assignment was very straightforward and covered the very basics of classes and objects in python. Being familiar with OOPS for sometime, I did not find any issues with the implementation. Although, I had to scratch my head a bit in Task 7 because of ambiguity surrounding test case in Piazza and requirement of Task7. It initially seemed that *net_population* function had to be deleted from the parent class. But after going through test case on Piazza, I realized that method overloading was needed.

5 Suggested Enhancements

Test case could be included as part of the assignment handout.