BME 646/ ECE695DL: Homework 1

Bianjiang Yang

January 18, 2022

1 Introduction

Homework1 covers some basics in programming using object oriented Python. Specifically, it includes creating class, instance variables, subclass, instance of a class, local variable, function in a class, initializing subclass using the 'super' operation, returning a function in a function. After implementing the code, I have a deeper understanding of the concept of 'callable', 'subclass' and basic coding style of OOP in Python3.

2 Methodology

Data: Provided in the assignment. No need to pre-process.
Tools: PyCharm for creating, editing and debugging the code. Python3 for running the code.
Techniques: Basic object oriented Python3 rules and concepts including: class, instance variables, subclass, instance of a class, local variable, function in a class.

3 Implementation and Results

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

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

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

    def density_calculator1(self):
```

```
        self.density = self.net_population() / self.area # density
        return self.density

    def density_calculator2(self):
        last_count = self.population[2] - self.population[0] + self
    .population[1] # correction
        self.current_net = self.population[2]
        self.population[2] = last_count
        self.density = self.current_net / self.area  # density
        return self.density

    def net_density(self, choice):
        if choice == 1:
            return self.density_calculator1
        elif choice == 2:
            if len(self.population) == 3:
                self.population.append(self.population[0] - self.
    population[1] + self.population[2])
                self.current_net = self.population[0] - self.
    population[1] + self.population[2]
            return self.density_calculator2
        else:
            raise ValueError('The \'choice\' Variable can only
    accept the value 1 or the value 2.')

    def net_population(self):
        if len(self.population) == 4:
            self.population[2] = self.population[3]
            self.population[3] = self.current_net
            self.current_net = self.population[0] - self.population
    [1] + (self.population[2] + self.population[3]) / 2
        if len(self.population) == 3:
            self.current_net = self.population[0] - self.population
    [1] + self.population[2]
        return self.current_net

def main():
    # obj_country = Countries("Piplipol", [40, 30, 20])
    obj = GeoCountry(capital="Polpip", population=[55, 10, 70],
    area=230)
    fn = obj.net_density(2)
    print("The density results for choice 2: ", fn())

    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())  # 1000
    ob1.net_density(2)
    print(ob1.population)  # [20,100,1080,1000]
    print(ob1.density)  # 200.0 (the value of density still uses
    the previous value of population)


if __name__ == "__main__":
    main()
```

Listing 1: Code

```
yangbj@qlabit:~$ python 695/Yang_Bianjiang_hw1.py
The density results for choice 2:  0.30434782608695654
0
[20, 100, 1000]
184.0
[20, 100, 1080]
200.0
0
8.33
[20, 100, 1080]
1000
[20, 100, 1080, 1000]
200.0
```

Figure 1: Printed Results

4 Lessons Learned

I learned how to create class, instance variables, subclass, instance of a class, local variable, function in a class, initialize subclass using the 'super' operation, return a function in a function. After implementing the code, I have a deeper understanding of the concept of 'callable', 'subclass' and basic coding style of OOP in Python3 such as inheriting the attributes.

5 Suggested Enhancements

I think we should not just be required to create some classes or functions. We should also try to run them to create more meaningful results. In that case, we can have a better understanding of these class and the OOP Python3.