BME646/ECE695DL: Homework 1

Spring 2022

Due Date: Monday, January 17,2022 (11:59pm ET)

1 Introduction

This homework covers some basics in programming using object oriented Python. The goal of this homework is to improve your understanding of the Python OO code in general, especially with regard to how it is used in PyTorch. This is the only homework you will get on general Python OO programming. Future homework assignments will be specific to using PyTorch classes directly or your own extensions of those classes for creating your DL solutions.

2 Goals

1. Handle classes and their inheritance.

2. Write a function that returns a custom function as output.

3 Background

3.1 Functions

Functions are treated as first-class objects in Python. They are allowed to accept one or more functions as arguments and return one or more functions. Just as:

```
def summation(nums):
    return  sum(nums)
def main(fun, args)
    result = fun(args)
    print(result)
if __name__ == "__main__":
    main(summation, [1,2,3])
```

3.2 Classes and their inheritance

Inheritance is a fundamental property of Python OO. It introduces us to its ability to obtain certain features (variables and methods) from its parent classes and make modifications, as well as come up with new ones. For example,

```python
class Person():
    def __init__(self, name):
        self.name = name
    def get_name(self):
        return self.name
class Employee(Person):
    def get_salary(self):
        return 1000
if __name__ == "__main__":
    ob1 = Employee("Ahmed")
    name = ob1.get_name()
    print(name)
```

4 Tasks

1. Create a class named `Countries` that has two instance variables named:
   - `capital`
   - `population`
2. Create an instance of your class (within `if __name__ == "__main__"`) and set `capital` to `Piplipol` and `population` to `[$40,30,20$]`. The `list` `population` represents the `[birth, death, last_count]` count (in the units of one thousand) of an instance of class `Countries` , in a given year. `last_count` denotes the net population from the immediate past year.
3. Expand your class `Countries` and define a new function `net_population()`. In this function, use the formula: `birth - death + last_count` to compute the local variable `current_net` and return it.
4. Extend your `Countries` class into a subclass named `GeoCountry`. Endow this class with two instance variables:
   - `area`

- density

5. Create an instance of this class GeoCountry (within if __name__ == "__main__"), and set capital to Polpip, population to [55,10,70], area to 230. Note: density will be calculated later and should not be passed as a function parameter.

6. Expand your class GeoCountry and define three new functions:

   - density_calculator1(): In this function, invoke net_population() in the parent class, to calculate and set the instance variable density as current_net / area.
   - density_calculator2(): While computing the population of the current year, an undetected bug resulted in last_count being replaced by current_net. In this function, you should correct this error in the instance variable population, and compute the new density with the corrected values.
   - net_density(): In this function, you must provide an argument variable choice in the function definition. This variable can only accept the values 1 or the value 2. If choice is set to 1 from the invoking instance, you must return the function density_calculator1() and if the choice is set to 2, then you must return the density_calculator2(). Please remember to return the function and not the outcome. You may verify this using the following code (within if __name__ == "__main__"):
     fn = obj.net_density(2)
     print(fn()) #where obj is your instance.

7. While working with this data, you realise that often births and deaths are not accurately reported. This leads to a margin of error, which skews your data. In order to overcome this you have three new modifications to incorporate:

   - Overwrite the parent class's function net_population(), in the child class.
   - Increase the size of the instance variable population by 1 and append the calculated current_net to it. The list population now is represented as [birth, death, second_last_count, last_count]. The new length of the instance variable population must be 4. second_last_count, now represents your prior last_count, and last_count is your prior current_net.
   - Modify the future calculations of current_net to: birth - death + (second_last_count + last_count) / 2.

5 Submission Instructions

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Compress your Python source code and pdf report(see the submission template released) into a singular zip file, naming it as your lastname_firstname.zip and upload it onto BrightSpace. **Your code must be your own work.**
- You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission.