ECE60146: Homework 1

Manish Kumar Krishne Gowda, 0033682812

(Spring 2024)

Introduction

In this assignment,implementation of Object-Oriented (OO) programming tasks were performed using Python. It is intended to serve as a foundational base for upcoming assignments, which will be specifically implemented using PyTorch.

Programming Task Execution and Outcomes

```
class Sequence(object):
  def __init__(self,array):
    self.array = array
    self.idx = -1 #start index for __next__
  def __call__(self):
    return self.array
  def __len__(self):
    return len(self.array)
  def __iter__(self): #Task 4 i.e. to use Sequence obj as iterator
    return(self)
  def __next__(self): #Task 4 i.e. to use Sequence obj as iterator
    self.idx += 1
    if self.idx == len(self.array):
      raise StopIteration
    else:
      return self.array[self.idx]
  def __eq__(self, other): #special method to compare two instances
      if len(self.array) == len(other.array):
        num_eq = 0
        for i in range(len(self.array)):
          if(self.array[i] == other.array[i]):
            num_eq += 1
        return num_eq #"num_eq" is count of elements that are identical at corresp
                                                    pos in two arrays.
      else:
        raise ValueError('Two arrays are not equal in length !')

class Arithmetic(Sequence): #Task 2 and 3 i.e. creating Arithmetic class extending
                                          Sequence
  def __init__(self,start,step):
    self.start = start
    self.step = step
    self.a_seq = []
  def __call__(self,length): #Task3 i.e. to make instances of Arithmetic class
                                          callable
    self.a_seq.clear() #clearing old seq to define a new sequence everytime the
                                          instance is called
    for i in range(length):
      self.a_seq.append(self.start + i*self.step) #AS = start + step*(n-1)  where
                                          n is the nth num in the seq
    super().__init__(self.a_seq)
    print(self.array)
```

```
class Geometric(Sequence): #Task 5 i.e. creating Geometric class extending
                                                    Sequence
  def __init__(self,start,ratio):
    self.start = start
    self.ratio = ratio
    self.g_seq = []
  def __call__(self,length): #Task5 i.e. to make instances of Geometric class
                                                      callable
    self.g_seq.clear()
    for i in range(length):
      self.g_seq.append(self.start * pow(self.ratio,i)) #GS = start*(ratio^(n-1))
                                                        where n is the nth num in the seq
    super().__init__(self.g_seq)
    print(self.array)
```

Task 1

A class named "Sequence" with an instance variable named array is created. Given parameter [0, 1, 2] is used for input and the reproduced results are shown in Task 1 output. Further outputs with a custom input of [2,4,6,8] is also shown in the code output Figure 1 . The input array is instantiated inside the __init__ function of Sequence class, while __call__ function makes the instance of this class (and hence the instances of its subclasses) callable.

```
#Test Task 1
ex_inst = Sequence(array = list(range(1,4))) #range(start,stop,step) => array = [1,2,3]
print(ex_inst())
```

```
[1, 2, 3]
```

```
#Custom Task 1
my_inst = Sequence(array = list(range(2,10,2))) #range(start,stop,step) => array = [2,4,6,8]
print(my_inst())
```

```
[2, 4, 6, 8]
```

Figure 1: Task 1 Output

Task 2, 3 & 4

"Arithmetic" class is derived from "Sequence" class and hence inherits its attributes. "__init__" method of the "Arithmetic" class overrides the "__init__" method of the "Sequence" class, defining the attributes "start" and "step" specific to the "Arithmetic" class (polymorphism). Similarly, "__call__" of "Arithmetic" class overrides the function of same name from the "Sequence" class, which defines a Arithmetic sequence for the given start, step and length values using the nth term of a AP series. The instance calling the "__call__" function of "Arithmetic" class further invokes the constructor of the "Sequence" superclass and initializes the "array" attribute of the instance.

"len(AS)" calls the "__len__" of the "Sequence" due to inheritance, which inturn returns the length of the "array" list attribute. The special functions "__iter__" and "__next__" renders the Sequence (and Arithmetic and Geometric objects) objects as iterators, enabling the printing of the array values inside the "print()" function.

Results of given test input and the custom input of own choice are shown in Figure 2

```
#Test Task 2,3 & 4
AS = Arithmetic(start=1, step=2)
AS(length=5)
print(len(AS))
print([n for n in AS])
```

```
[1, 3, 5, 7, 9]
5
[1, 3, 5, 7, 9]
```

```
[65] #Custom Task 2,3 & 4
     AS = Arithmetic(start=2, step=3)
     AS(length=4)
     print(len(AS)) #4
     print([n for n in AS])
```

```
[2, 5, 8, 11]
4
[2, 5, 8, 11]
```

Figure 2: Task 2,3 & 4 Output

Task 5

"Geometric" class extends "Sequence" superclass similar to the "Arithmetic" class. Similar to the "Arithmetic" class, the "Geometric" class inherits all the attributes and methods of the "Sequence" class. Further, the nth term of a GP is used to define the sequence given the "start", common "ratio" and the "length" of the series. Test input of a length 8 sequence and a custom test input of length 4 sequence were tested for the validity of the code and is shown in Figure 3

Task 6

Special (dunder) method "__eq__" is called when two objects of "Sequence" type are called (objects of "Arithmetic" type and "Geometric" type are also objects of "Sequence" type due to inheritance). This function compares the "array" attribute of the two objects element by element and returns the total number of matching elements in corresponding positions of the array. In other words, the comparison (A==B) between two sequences A and B is performed element-wise based on their corresponding positions in "array" attribute. The comparison check will test if the first element of A is equal to the first element of B, the second element of A is equal to the second element of B, and so forth and eventually return the total number matches in the "array" comparison. Further, if the length of the "array" attribute of the two objects undergoing comparison are not the same, a "ValueError" exception is raised as per the task instructions. The results of the provided test input and a custom test input is shown in Figure 4 and Figure 5 respectively.

```
#Test Task 5
GS = Geometric(start=1, ratio=2)
GS(length=8)
print(len(GS)) #8
print([n for n in GS])
```

```
[1, 2, 4, 8, 16, 32, 64, 128]
8
[1, 2, 4, 8, 16, 32, 64, 128]
```

```
[67] #Custom Task 5
     GS = Geometric(start=2, ratio=3)
     GS(length=4)
     print(len(GS))
     print([n for n in GS])
```

```
[2, 6, 18, 54]
4
[2, 6, 18, 54]
```

Figure 3: Task 5 Output

Conclusion

The Given Programming assignment was successfully implemented and the results were verified for the given test inputs and a custom input of own choice. The tasks helped to provide a overview of the OOP concepts in python.

```
#Test Task 6
AS = Arithmetic(start=1, step=2)
AS(length=5)
GS = Geometric(start=1, ratio=2)
GS(length=5)
print(AS == GS)
#change GS instance length to raise ValueError
GS(length=8)
print(AS == GS)
```

```
[1, 3, 5, 7, 9]
[1, 2, 4, 8, 16]
1
[1, 2, 4, 8, 16, 32, 64, 128]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-68-1c40b447312f> in <cell line: 9>()
      7 #change GS instance length to raise ValueError
      8 GS(length=8)
----> 9 print(AS == GS)

<ipython-input-61-e29c451fbaea> in __eq__(self, other)
     23         return num_eq #"num_eq" is count of elements that are identical at corresponding positions in two arrays.
     24       else:
---> 25         raise ValueError('Two arrays are not equal in length !')
     26
     27 class Arithmetic(Sequence): #Task 2 and 3 i.e. creating Arithmetic class extending Sequence

ValueError: Two arrays are not equal in length !
```

Figure 4: Task 6 Output for given Test Input

```
#Custom Task 6
AS = Arithmetic(start=0, step=3)
AS(length=10)
GS = Geometric(start=1, ratio=4)
GS(length=10)
print(AS == GS)
#change GS instance length to raise ValueError
GS(length=5)
print(AS == GS)
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
[1, 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262144]
0
[1, 4, 16, 64, 256]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-69-5355d5173fba> in <cell line: 9>()
      7 #change GS instance length to raise ValueError
      8 GS(length=5)
----> 9 print(AS == GS)

<ipython-input-61-e29c451fbaea> in __eq__(self, other)
     23         return num_eq #"num_eq" is count of elements that are identical at corresponding positions in two arrays.
     24       else:
---> 25         raise ValueError('Two arrays are not equal in length !')
     26
     27 class Arithmetic(Sequence): #Task 2 and 3 i.e. creating Arithmetic class extending Sequence

ValueError: Two arrays are not equal in length !
```

Figure 5: Task 6 Output for custom Test Input