HW 1: Object Oriented Programming

David E. Farache, Email ID: dfarache@purdue.edu

January 16, 2023

1 Introduction

The homework focuses upon the principles of object oriented programming in python including the basic of creating objects, initializing their attributes, and extended one class to the others. Furthermore, python specific methods were called "magic methods" were used.

2 Methodology

For this work, three class were created called Sequence, Fibonacci, and Prime. Sequence was a base class that was extended to the Fibonacci and Prime classes that had an array attribute. The Fibonacci class functioned by taking input of two integers that would then be used to return an array of input length l and using equation 1 to fill out the array. Whereby f_{n-1} would be second_value and f_{n-2} would be first_value.

$$f_n = f_{n-1} + f_{n-2} \tag{1}$$

The Prime class had no input required to initialize that class. The Prime class also returned an array with length l but instead of Fibonacci numbers it returns prime numbers starting at 2. It did so by determining the value prime of the modulus returned non zero values when dividing by the existing list of prime numbers. Both the Prime and Fibonacci classes performed this tasks via the _call_ method that activates a method when the class is called with the required input given. Each function was made so that one could iterate through the array produce via the _iter_ method and the length of the array was returned via the _len_ method which enables one to use the len() operator upon the class to return a value.

The _gt_ method was later added enabling the use of the > operator to compare Prime and Fibonacci created arrays to return the number of values between individual elements in the same index. A line ValueError, enables the catching and returning of an error system that further implemented if lengths of both array were no equivalent.

3 Implementation and Results

3.1 Task 1

Create a class called Sequence with initialization of an array as an attribute.

```
class Sequence(object):
    def __init__(self, array):
        self.array = array
```

Listing 1: *Create Sequence class*

3.2 Task 2

Create a class called Fibonacci with initialization of integer values of first_value and second_value.

```
class Fibonacci(Sequence):
    def __init__(self, first_value, second_value):
      self.first_value = first_value
      self.second_value = second_value
```

Listing 2: *Create Fibonacci class extending Sequence to it*

3.3 Task 3

Using the _call_ method, have the it when the class is called, produce an array of input length of values using the Fibonacci sequence. The initial and second value of the array should be from attributes of first_value and second_value.

```
class Fibonacci(Sequence):
    def __init__(self, first_value, second_value):
      self.first_value = first_value
      self.second_value = second_value

    def __call__(self, length):
      hold_array = []
      hold_array = [self.first_value, self.second_value]
      for i in range(0, length-2):
        val = hold_array[i] + hold_array[i+1]
        hold_array.append(val)
      self.array = hold_array
      return self.array
```

```
FS = Fibonacci ( first_value =1 , second_value =2 )
FS ( length =5 )
```

Listing 3: *Return array of Fibonacci number given two initial values*

The picture was taken form the code to prove result desired was output-ed.

Figure 1: *Output array of Fibonacci values of length 5*

```
[1, 2, 3, 5, 8]
```

3.4 Task 4

Modify the Fibonacci class so that the len() operator returns the length of the array when applied on an instance of the class. Furthermore, have the class that iterates through the array created, returning each element in the array when looping through using the _iter_ function

```
class Sequence(object):
  def __init__(self, array):
    self.array = array

  def get_number(self, pos):
    return self.array[pos]

  def __iter__(self):
      return (n for n in self.array)

class Fibonacci(Sequence):
    def __init__(self, first_value, second_value):
      self.first_value = first_value
      self.second_value = second_value

    def __call__(self, length):
      hold_array = []
      hold_array = [self.first_value, self.second_value]
      for i in range(0, length-2):
        val = hold_array[i] + hold_array[i+1]
        hold_array.append(val)
      self.array = hold_array
      return self.array

    def __len__(self):
      return len(self.array)
```

```
FS = Fibonacci ( first_value =1 , second_value =2 )
print(FS ( length =5 ))
print(len( FS ))
print([n for n in FS])
```

Listing 4: *Add operator for length function and iteration of object*

The picture was taken form the code to prove result desired was output-ed.

Figure 2: *Output of length of array and enabling iteration of array*

```
[1, 2, 3, 5, 8]
5
[1, 2, 3, 5, 8]
```

3.5 Task 5

Create a new class called Prime that is identical the Fibonacci class except it returns an array of prime number when called with length l and does not require the original two inputs.

```
class Prime(Sequence):
    def __init__(self):
      self.value = 2

    def __call__(self, length):
      self.value = 2
      self.array = [2]
```

```
      while len(self.array) < length:
        self.value = self.value + 1
        flag = True

        for i in self.array:
          if(self.value % i == 0):
            flag = False

        if(flag == True):
          self.array.append(self.value)

      return self.array

    def __len__(self):
      return len(self.array)
```

```
PS = Prime ()
PS ( length =8 )
print(PS ( length =8 ))
print (len( PS ) )
print ([n for n in PS])
```

Listing 5: *Prime class creation*

The picture was taken from the code to prove the result desired was output-ed.

Figure 3: *Output of Prime class of length 8*

```
[2, 3, 5, 7, 11, 13, 17, 19]
8
[2, 3, 5, 7, 11, 13, 17, 19]
```

3.6 Task 6

Modify the base class Sequence such that the operator > applied to extended classes of sequence will return the amount of element values of the same index or greater for the compared object's array. If the arrays are not equivalent lengths of the objects being compared then use ValueError exception to throw an error.

```
class Sequence(object):
  def __init__(self, array):
    self.array = array

  def get_number(self, pos):
    return self.array[pos]

  def __iter__(self):
    return (n for n in self.array)

  def __gt__(self, other):
    if(len(self.array) != len(other.array)):
      raise ValueError("Two arrays are not equal in length!")

```

```
    num_greater = 0
    for i, j in zip(self.array, other.array):
      if(i > j):
        num_greater += 1

    return num_greater


class Fibonacci(Sequence):
    def __init__(self, first_value, second_value):
      self.first_value = first_value
      self.second_value = second_value

    def __call__(self, length):
      hold_array = []
      hold_array = [self.first_value, self.second_value]
      for i in range(0, length-2):
        val = hold_array[i] + hold_array[i+1]
        hold_array.append(val)
      self.array = hold_array
      return self.array

    def __len__(self):
      return len(self.array)

class Prime(Sequence):
    def __init__(self):
      self.value = 2

    def __call__(self, length):
      self.value = 2
      self.array = [2]
      while len(self.array) < length:
        self.value = self.value + 1
        flag = True

        for i in self.array:
          if(self.value % i == 0):
            flag = False

        if(flag == True):
          self.array.append(self.value)

      return self.array

    def __len__(self):
      return len(self.array)
```

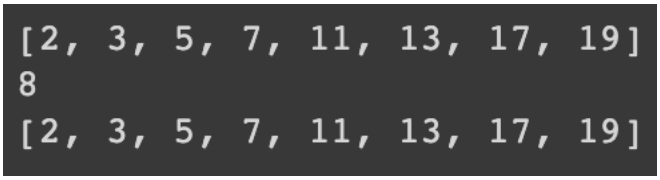
```
FS = Fibonacci ( first_value =1 , second_value =2 )
FS ( length =8 )
PS = Prime ()
PS ( length =8 )
print(FS ( length =8 ) )
print(PS ( length =8 ))
print( FS > PS )
PS ( length =5 )
```

```
9 print( FS > PS )
```

Listing 6: *Add comparibility between array elemental values to determine amount of greater element to its index*

The picture was taken from the code to prove the result desired was output-ed.

Figure 4: *Output of Prime class of length 8*



4 Lessons learned

The majority of this assignment was review as I have worked in object oriented programming previously but that was in java, so learning the distinct properties within the python library has been useful to learn and practice with.