PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Deep Learning

Homework 1

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: January 16, 2023

1 Theory

In this homework, we work on general Python Object Oriented Programming. It is a programming approach that utilizes the following concepts:

1. **Class** : It is an entity that consists of certain attributes that can be referred to by the same name.
2. **Inheritance**: It is the property of a class to derive its properties from another class.
3. **Polymorphism**: It is the property of an object to be interpreted in different ways.
4. **Encapsulation**: It is the act of wrapping up data and the functions associated with it.
5. **Abstraction**: It is the property of hiding unnecessary implementation details from the user.

For this homework, I have referred to Prof Kak's tutorial on OOP in Python.

2 Experiment

2.1 Task 1

The first task is to create a class called *Sequence*, as shown below.

```
In [2]: class Sequence():
            def __init__(self,array):
                self.array = array
```

Figure 1:

2.2 Task 2

Now we create a subclass of *Sequence* called *Fibonacci* which consists of a *__init__*() method having two parameters (which serve as the first two members of the Fibonacci series).

```
In [3]: class Fibonacci(Sequence):
            def __init__(self,first_value,second_value):
                super().__init__([])
                self.first_value = first_value
                self.second_value = second_value
```

Figure 2:

2.3 Task 3

Now we make the instances of *Fibonacci* callable by implementing the *__call__*() method. This method calls the *series*() method to compute the Fibonacci series.

In [3]:
```
class Fibonacci(Sequence):
    def __init__(self,first_value,second_value):
        super().__init__([])
        self.first_value = first_value
        self.second_value = second_value

    def series(self):
        t1 = self.first_value
        t2 = self.second_value
        self.array = [t1,t2]
        for i in range(self.length-2):
            t3 = t2
            t2 = t1+t2
            t1 = t3
            self.array.append(t2)

    def __call__(self,length):
        self.length = length
        self.array =[]
        self.series()
        print(self.array)
```

Figure 3:

2.4 Task 4

Then we make the instances of *Sequence* iterable by implementing the *__iter__*() method which calls the iterator class *Sequence_iter*. This class consists of the *__iter__*(), *__next_*() and *__init__*() methods.

In [2]:
```
class Sequence():
    def __init__(self,array):
        self.array = array

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

    def __iter__(self):
        return Sequence_iter(self)

class Sequence_iter():
    def __init__(self,obj):
        self.items = obj.array
        self.index = -1

    def __iter__(self):
        return self

    def __next__(self):
        self.index= self.index+1
        if self.index<len(self.items):
            return self.items[self.index]
        else:
            raise StopIteration
```

Figure 4:

2.5 Task 5

Now we create another subclass of *Sequence* called *Prime* which has the same attributes as *Fibonacci* apart from the two parameters in the *__init__*() method.

In [4]:
```
class Prime(Sequence):
    def __init__(self):
        super().__init__([])

    def series(self):
        i = 2
        while(len(self.array)<self.length):
            flag=0;
            for j in range(2,int(m.sqrt(i))+1):
                if i%j==0:
                    flag = 1
                    break
            if flag == 0:
                self.array.append(i)
            i = i+1

    def __call__(self,length):
        self.length = length
        self.array =[]
        self.series()
        print(self.array)
```

Figure 5:

2.6 Task 6

Finally, we modify *Sequence* such that two instances of the same length can be compared with the same length. We achieve this by implementing the $\_\_gt\_\_()$ method, which is called operator overloading.

In [2]:
```
class Sequence():
    def __init__(self,array):
        self.array = array

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

    def __iter__(self):
        return Sequence_iter(self)

    def __gt__(self,other):
        if len(self.array)!=len(other.array):
            raise ValueError("Two arrays are not equal in length!")
        else:
            count = 0
            for i in range(len(self.array)):
                if self.array[i]>other.array[i]:
                    count = count+1

            return count

class Sequence_iter():
    def __init__(self,obj):
        self.items = obj.array
        self.index = -1

    def __iter__(self):
        return self

    def __next__(self):
        self.index= self.index+1
        if self.index<len(self.items):
            return self.items[self.index]
        else:
            raise StopIteration
```

Figure 6:

3 Results-1

In this section, we reproduce the results of the document provided.

3.1 Task 3

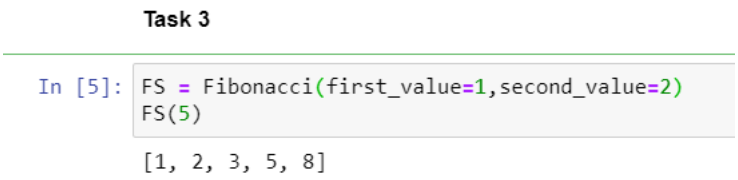

Figure 7: Output

3.2 Task 4

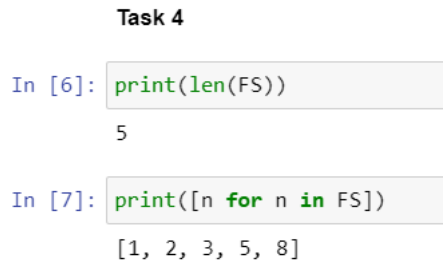

Figure 8: Output

3.3 Task 5

Task 5

In [8]:
```
#Task 5
PS = Prime()
PS(8)
```

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

In [9]:
```
print(len(PS))
```

```
8
```

In [10]:
```
print([n for n in PS])
```

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

Figure 9: Output

3.4 Task 6

Task 6

In [11]:
```
FS = Fibonacci(first_value=1,second_value=2)
FS(8)
```

```
[1, 2, 3, 5, 8, 13, 21, 34]
```

In [12]:
```
PS = Prime()
PS(8)
```

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

In [13]:
```
print(FS>PS)
```

```
2
```

In [14]:
```
PS(5)
```

```
[2, 3, 5, 7, 11]
```

In [15]:
```
print(FS>PS)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [15], in <cell line: 1>()
----> 1 print(FS>PS)

Input In [2], in Sequence.__gt__(self, other)
      5 def __gt__(self,other):
      6     if len(self.array)!=len(other.array):
----> 7         raise ValueError("Two arrays are not equal in length!")
      8     else:
      9         count = 0

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

Figure 10: Output

4 Results-2

In this section, we produce our own results.

4.1 Task 3

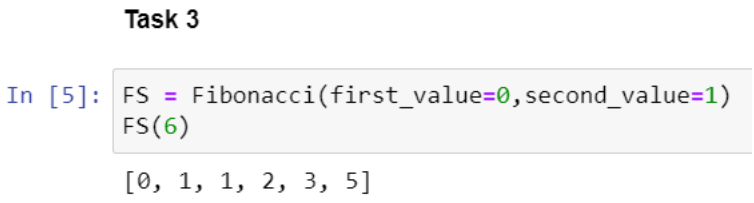

Figure 11: Output

4.2 Task 4

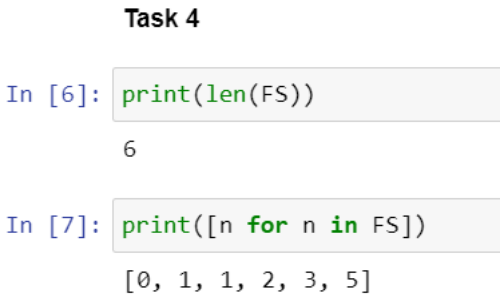

Figure 12: Output

4.3 Task 5

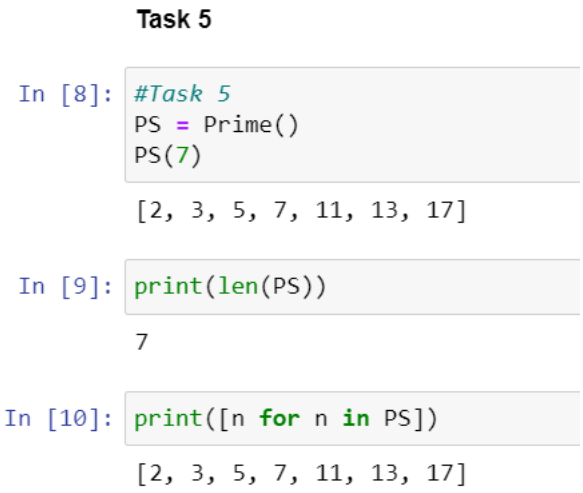

Figure 13: Output

4.4 Task 6

Task 6

In [11]:
```
FS = Fibonacci(first_value=0,second_value=1)
FS(12)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

In [12]:
```
PS = Prime()
PS(12)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

In [13]:
```
print(FS>PS)
```

```
3
```

In [14]:
```
PS(11)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

In [15]:
```
print(FS>PS)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [15], in <cell line: 1>()
----> 1 print(FS>PS)

Input In [2], in Sequence.__gt__(self, other)
     11 def __gt__(self,other):
     12     if len(self.array)!=len(other.array):
---> 13         raise ValueError("Two arrays are not equal in length!")
     14     else:
     15         count = 0

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

Figure 14: Output

In [1]:

```
import math as m
```

In [2]:

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

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

    def __iter__(self):
        return Sequence_iter(self)

    def __gt__(self,other):
        if len(self.array)!=len(other.array):
            raise ValueError("Two arrays are not equal in length!")
        else:
            count = 0
            for i in range(len(self.array)):
                if self.array[i]>other.array[i]:
                    count = count+1

            return count

class Sequence_iter():
    def __init__(self,obj):
        self.items = obj.array
        self.index = -1

    def __iter__(self):
        return self

    def __next__(self):
        self.index= self.index+1
        if self.index<len(self.items):
            return self.items[self.index]
        else:
            raise StopIteration
```

In [3]:

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

    def series(self):
        t1 = self.first_value
        t2 = self.second_value
        self.array = [t1,t2]
        for i in range(self.length-2):
            t3 = t2
            t2 = t1+t2
            t1 = t3
            self.array.append(t2)

    def __call__(self,length):
        self.length = length
        self.array =[]
        self.series()
        print(self.array)
```

In [4]:

```
class Prime(Sequence):
    def __init__(self):
        super().__init__([])

    def series(self):
        i = 2
        while(len(self.array)<self.length):
            flag=0;
            for j in range(2,int(m.sqrt(i))+1):
                if i%j==0:
                    flag = 1
                    break
            if flag == 0:
                self.array.append(i)
            i = i+1

    def __call__(self,length):
        self.length = length
        self.array =[]
        self.series()
        print(self.array)
```

Task 3

In [5]:

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

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

Task 4

In [6]:

```
print(len(FS))
```

```
5
```

In [7]:

```
print([n for n in FS])
```

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

Task 5

In [8]:

```
#Task 5
PS = Prime()
PS(8)
```

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

In [9]:

```
print(len(PS))
```

```
8
```

In [10]:

```
print([n for n in PS])
```

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

Task 6

In [11]:

```
FS = Fibonacci(first_value=1,second_value=2)
FS(8)
```

```
[1, 2, 3, 5, 8, 13, 21, 34]
```

In [12]:

```
PS = Prime()
PS(8)
```

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

In [13]:

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

```
2
```

In [14]:

```
PS(5)
```

```
[2, 3, 5, 7, 11]
```

In [15]:

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [15], in <cell line: 1>()
----> 1 print(FS>PS)

Input In [2], in Sequence.__gt__(self, other)
     11 def __gt__(self,other):
     12     if len(self.array)!=len(other.array):
---> 13         raise ValueError("Two arrays are not equal in length!")
     14     else:
     15         count = 0

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

In [ ]:

In [1]:

```
import math as m
```

In [2]:

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

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

    def __iter__(self):
        return Sequence_iter(self)

    def __gt__(self,other):
        if len(self.array)!=len(other.array):
            raise ValueError("Two arrays are not equal in length!")
        else:
            count = 0
            for i in range(len(self.array)):
                if self.array[i]>other.array[i]:
                    count = count+1

            return count

class Sequence_iter():
    def __init__(self,obj):
        self.items = obj.array
        self.index = -1

    def __iter__(self):
        return self

    def __next__(self):
        self.index= self.index+1
        if self.index<len(self.items):
            return self.items[self.index]
        else:
            raise StopIteration
```

In [3]:

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

    def series(self):
        t1 = self.first_value
        t2 = self.second_value
        self.array = [t1,t2]
        for i in range(self.length-2):
            t3 = t2
            t2 = t1+t2
            t1 = t3
            self.array.append(t2)

    def __call__(self,length):
        self.length = length
        self.array =[]
        self.series()
        print(self.array)
```

In [4]:

```
class Prime(Sequence):
    def __init__(self):
        super().__init__([])

    def series(self):
        i = 2
        while(len(self.array)<self.length):
            flag=0;
            for j in range(2,int(m.sqrt(i))+1):
                if i%j==0:
                    flag = 1
                    break
            if flag == 0:
                self.array.append(i)
            i = i+1

    def __call__(self,length):
        self.length = length
        self.array =[]
        self.series()
        print(self.array)
```

Task 3

In [5]:

```
FS = Fibonacci(first_value=0,second_value=1)
FS(6)
```

```
[0, 1, 1, 2, 3, 5]
```

Task 4

In [6]:

```
print(len(FS))
```

```
6
```

In [7]:

```
print([n for n in FS])
```

```
[0, 1, 1, 2, 3, 5]
```

Task 5

In [8]:

```
#Task 5
PS = Prime()
PS(7)
```

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

In [9]:

```
print(len(PS))
```

```
7
```

In [10]:

```
print([n for n in PS])
```

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

Task 6

In [11]:

```
FS = Fibonacci(first_value=0,second_value=1)
FS(12)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

In [12]:

```
PS = Prime()
PS(12)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

In [13]:

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

```
3
```

In [14]:

```
PS(11)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

In [15]:

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [15], in <cell line: 1>()
----> 1 print(FS>PS)

Input In [2], in Sequence.__gt__(self, other)
     11 def __gt__(self,other):
     12     if len(self.array)!=len(other.array):
---> 13         raise ValueError("Two arrays are not equal in length!")
     14     else:
     15         count = 0

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

In [ ]:

```

```