# hw1_Alan_Chen

## 1. Implementation Description and Outputs With the Given Parameters

### Task 2.1

Create a class called BioModel, which will be base class for  many subclasses in this assignment. The input of this class is expected to be a list of numbers, and the list is then stored as an instance variable, self.sequence.

```
class BioModel(object):
    def __init__(self, sequence):
        self.sequence = sequence
```

### Task 2.2

Create a class called ExponentialGrowthModel, which is a subclass of BioModel. When we do initialization, remember to initialize its superclass first, in this case, we call super() function to achieve that. And we also initiate self.start and rate for further use.

```
class ExponentialGrowthModel(BioModel):
    def __init__(self, start, rate):
        # remember to init superclass as well!!! super(SubclassName, self).__ini
        super().__init__([]) #initialize BioModel with empty sequence

        self.start = start
        self.rate = rate
```

### Task 2.3

Expand ExponentialGrowthModel to make it callable which is called with a parameter named length, and return a sequence based on “next value = current value × (1 +rate)”. We can acheieve these by using ”**__call__”** and “**__len__”** functions, in the call function, I use a for loop to append the sequence with the formula mentioned above.

In addition, calling the instance of ExponentialGrowthModel also prints out the sequence

```
# Task 2.2
class ExponentialGrowthModel(BioModel):
    def __init__(self, start, rate):
        # remember to init superclass as well!!! super(SubclassName, self).__ini
        super().__init__([]) #initialize BioModel with empty sequence
```

```
        self.start = start
        self.rate = rate


    # Task 2.3
    def __call__(self, length):
        self.sequence = [self.start]
        for i in range(length - 1):
            self.sequence.append(self.sequence[i] * (1+self.rate))

        print(self.sequence)
        return self.sequence

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

### Output with Given Parameters:

```
GM = ExponentialGrowthModel(start=100, rate=0.1)
GM(length=5) # [100, 110, 121, 133.1, 146.41]
print(len(GM))
```

Output:

```
[100, 110.00000000000001, 121.00000000000003, 133.10000000000005, 146.4100000000
5
```

## Task 2.4

By using “__iter__” and “__next__”, we can make the class iterable. By setting the index to negative 1 and +1 in every iteration, we will be able to keep track of the current index. While we are not exceeding the last index, we return the current number. And StopIteration when it’s out of the last index.

```
#  Task 2.1
class BioModel(object):
    def __init__(self, sequence):
        self.sequence = sequence


    # Task 2.4
    def __iter__(self):
        # Initialize the index
```

```
        self.index =-1

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

### Output with Given Parameters:

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

Output:

```
[100, 110.00000000000001, 121.00000000000003, 133.10000000000005, 146.4100000000
```

## Task 2.5

Taks 2.5 is basically the same thing as ExponentialGrowthModel, the only difference is the math function, the function is " next value = current value × (1 −rate)". So what I did was copy ExponentialGrowthModel and change the function while maintaining other logic.

```
# Task 2.5
class ExponentialDecayModel(BioModel):
    def __init__(self, start, rate):
        # init superclass
        super().__init__([])

        self.start = start
        self.rate = rate

    def __call__(self, length):
        self.sequence = [self.start]

        for i in range(length - 1):
            self.sequence.append(self.sequence[i] * (1-self.rate))

        print(self.sequence)
        return self.sequence
```

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

### Output with Given Parameters:

```
DM = ExponentialDecayModel(start=100, rate=0.2)
DM(length=5) # [100, 80, 64, 51.2, 40.96]
print(len(DM)) # 5
print([n for n in DM]) # [100, 80, 64, 51.2, 40.96]
```

Output:

```
[100, 80.0, 64.0, 51.2, 40.960000000000001]
5
[100, 80.0, 64.0, 51.2, 40.960000000000001]
```

## Task 2.6

To make two instances able to use the == operator, we use “__eq__”. Use an if function to check if two sequences have the same length. If so, return the number of matching elements using the sum function. If not, raise an value error with error message.

```
#  Task 2.1
class BioModel(object):
    def __init__(self, sequence):
        self.sequence = sequence


    # Task 2.4
    def __iter__(self):
        # Initialize the index
        self.index =-1

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


    # Task 2.6
    def __eq__(self, other):
```

```
        if len(self.sequence) == len(other.sequence):
            return sum(1 for a, b in zip(self.sequence, other.sequence) if a ==
        else:
            raise ValueError("Two arrays are not equal in length!")
```

### Output with Given Parameters:

```
GM = ExponentialGrowthModel(start=100, rate=0.1)
GM(length=5) # [100, 110, 121, 133.1, 146.41]

GM2 = ExponentialGrowthModel(start=100, rate=0.2)
GM2(length=5) # [100, 120, 144, 172.8, 207.36]

print(GM == GM2) # 1


GM3 = ExponentialGrowthModel(start=100, rate=0.2)
GM3(length=3) # [100, 120, 144]

print(GM == GM3) # will raise an error
```

Output:

```
[100, 110.00000000000001, 121.00000000000003, 133.10000000000005, 146.4100000000
[100, 120.0, 144.0, 172.79999999999998, 207.35999999999999]
1
[100, 120.0, 144.0]

          ^^^^^^^^^
  File "d:\MS Purdue\1.5\ECE60146\HW1\HW1.py", line 29, in __eq__
    raise ValueError("Two arrays are not equal in length!")
ValueError: Two arrays are not equal in length!
```

## Task 3.1

The CombinedBioModel class is a subclass of BioModel that combines both models we created, we need to initialize BioModel as usual, and I initialize both models in “__init__” as well. When called with a length parameter, it generate a sequence for both models and does the multiplication elementwise, then stores and returns the combined sequence.

```
class CombinedBioModel(BioModel):
    def __init__(self, growth_start, growth_rate, decay_start, decay_rate):
        super().__init__([])
        self.growth = ExponentialGrowthModel(growth_start, growth_rate)
```

```
        self.decay = ExponentialDecayModel(decay_start, decay_rate)

    def __call__(self, length):
        self.growth(length=length)
        self.decay(length=length)
        self.sequence = []
        for i in range(len(self.growth)):
            self.sequence.append(self.growth.sequence[i] * self.decay.sequence[i
        print(self.sequence)
        return self.sequence
```

### Output with Given Parameters:

```
CBM = CombinedBioModel(growth_start=100, growth_rate=0.1, decay_start=1.0, decay
CBM(length=5) # [100.0, 104.50, 109.20, 114.12, 119.25]
```

Output:

The first one is the growth sequence, the second one is the decay sequence, and the their one is the combined sequence.

```
[100, 110.00000000000001, 121.00000000000003, 133.10000000000005, 146.4100000000
[1.0, 0.95, 0.9025, 0.8573749999999999, 0.8145062499999999]
[100.0, 104.50000000000001, 109.20250000000003, 114.11661250000003, 119.25186006
```

## Task 3.2

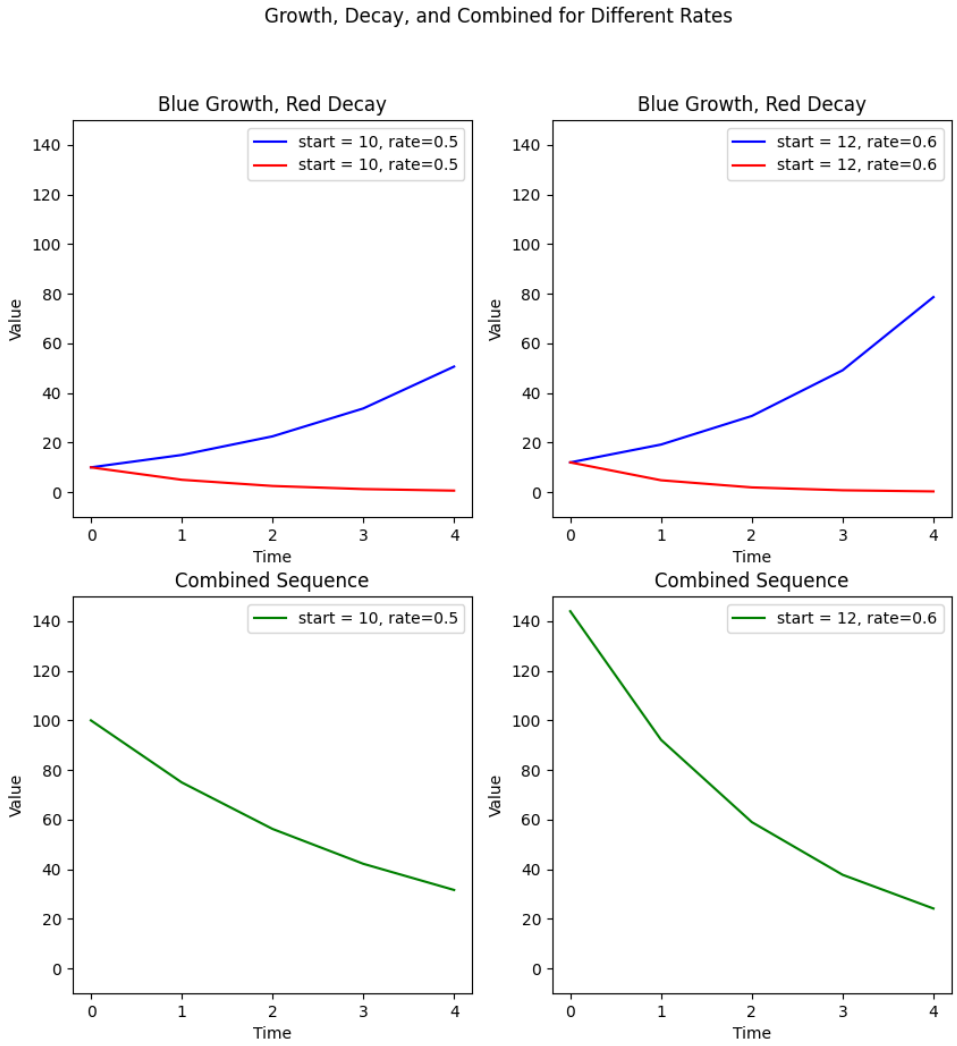
Created a visualization comparing growth, decay, and combined model with two sets of parameters. THe first part of the code is creating the six model that we are going to compare. The second part of code is creating the plot, we are using matplotlib in this assignments, we can use it to create 4 subplot that enable us  to easily compare those outcomes. In my perspective, I found out that when the difference of growth model and decay model went bigger, the starting value of combined model also grows simultaneously .

```
# Task 3.2
start1 = 10
start2 = 12
rate1 = 0.5
rate2 = 0.6
GM1 = ExponentialGrowthModel(start=start1, rate=rate1)
GM1(length=5)
DM1 = ExponentialDecayModel(start=start1, rate=rate1)
DM1(length=5)
CM1 = CombinedBioModel(growth_start=start1, growth_rate=rate1, decay_start=start
```

```
CM1(length=5)
GM2 = ExponentialGrowthModel(start=start2, rate=rate2)
GM2(length=5)
DM2 = ExponentialDecayModel(start=start2, rate=rate2)
DM2(length=5)
CM2 = CombinedBioModel(growth_start=start2, growth_rate=rate2, decay_start=start
CM2(length=5)

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

axs[0, 0].plot(GM1.sequence, color="blue", label="start = 10, rate=0.5")
axs[0, 0].plot(DM1.sequence, color="red", label="start = 10, rate=0.5")
axs[0, 0].set_title("Blue Growth, Red Decay")
axs[0, 0].set_xlabel('Time')
axs[0, 0].set_ylabel('Value')
axs[0, 0].legend()
axs[0, 0].set_ylim(-10, 150)

axs[0, 1].plot(GM2.sequence, color="blue", label="start = 12, rate=0.6")
axs[0, 1].plot(DM2.sequence, color="red", label="start = 12, rate=0.6")
axs[0, 1].set_title("Blue Growth, Red Decay")
axs[0, 1].set_xlabel('Time')
axs[0, 1].set_ylabel('Value')
axs[0, 1].legend()
axs[0, 1].set_ylim(-10, 150)

axs[1, 0].plot(CM1.sequence, color="green", label="start = 10, rate=0.5")
axs[1, 0].set_title("Combined Sequence")
axs[1, 0].set_xlabel('Time')
axs[1, 0].set_ylabel('Value')
axs[1, 0].legend()
axs[1, 0].set_ylim(-10, 150)

axs[1, 1].plot(CM2.sequence, color="green", label="start = 12, rate=0.6")
axs[1, 1].set_title("Combined Sequence")
axs[1, 1].set_xlabel('Time')
axs[1, 1].set_ylabel('Value')
axs[1, 1].legend()
axs[1, 1].set_ylim(-10, 150)

plt.suptitle("Growth, Decay, and Combined for Different Rates")
plt.legend()
```

```
plt.show()
```

Output:




```
[10, 15.0, 22.5, 33.75, 50.625]
[10, 5.0, 2.5, 1.25, 0.625]
[10, 15.0, 22.5, 33.75, 50.625]
[10, 5.0, 2.5, 1.25, 0.625]
[100, 75.0, 56.25, 42.1875, 31.640625]
[12, 19.200000000000003, 30.720000000000006, 49.152000000000015, 78.643200000000
[12, 4.800000000000001, 1.9200000000000004, 0.7680000000000002, 0.307200000000000
[12, 19.200000000000003, 30.720000000000006, 49.152000000000015, 78.643200000000
```

```
[12, 4.800000000000001, 1.9200000000000004, 0.7680000000000002, 0.30720000000000
[144, 92.16000000000003, 58.98240000000002, 37.74873600000002, 24.15919104000002
```

# 2. Input Parameters of My Choice

## Task 2.7

### Input Parameters of My Choice

```
# Task 2.7
GM = ExponentialGrowthModel(start=200, rate=0.9)
GM(length=5)
GM2 = ExponentialDecayModel(start=400, rate=0.1)
GM2(length=5)
print(len(GM)) # 5
print([n for n in GM])
print(len(GM2)) # 5
print([n for n in GM2])
print(GM == GM2) # 0
print(GM == GM) # 5

GM3 = ExponentialDecayModel(start=100, rate=0.2)
GM3(length=3)
# print(GM == GM3) # will raise an error
```

Output:

```
200, 380.0, 722.0, 1371.8, 2606.4199999999996]
[400, 360.0, 324.0, 291.6, 262.44000000000005]
5
[200, 380.0, 722.0, 1371.8, 2606.4199999999996]
5
[400, 360.0, 324.0, 291.6, 262.44000000000005]
0
5
[100, 80.0, 64.0]
```

# 3. Source Code

```
import matplotlib.pyplot as plt

print("---------------------- Task 2.1 ----------------------")
```

```
#  Task 2.1
class BioModel(object):
    def __init__(self, sequence):
        self.sequence = sequence


    # Task 2.4
    def __iter__(self):
        # Initialize the index
        self.index =-1

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


    # Task 2.6
    def __eq__(self, other):
        if len(self.sequence) == len(other.sequence):
            return sum(1 for a, b in zip(self.sequence, other.sequence) if a ==
        else:
            raise ValueError("Two arrays are not equal in length!")

print("----------------------- Task 2.2 -----------------------")
# Task 2.2
class ExponentialGrowthModel(BioModel):
    def __init__(self, start, rate):
        # remember to init superclass as well!!! super(SubclassName, self).__ini
        super().__init__([]) #initialize BioModel with empty sequence

        self.start = start
        self.rate = rate


    # Task 2.3
    def __call__(self, length):
        self.sequence = [self.start]
        for i in range(length - 1):
            self.sequence.append(self.sequence[i] * (1+self.rate))

        print(self.sequence)
```

```
        return self.sequence

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

GM = ExponentialGrowthModel(start=100, rate=0.1)
GM(length=5) # [100, 110, 121, 133.1, 146.41]
print(len(GM))

print("----------------------- Task 2.4 -----------------------")
# Task 2.4
print([n for n in GM])

print("----------------------- Task 2.5 -----------------------")
# Task 2.5
class ExponentialDecayModel(BioModel):
    def __init__(self, start, rate):
        # init superclass
        super().__init__([])

        self.start = start
        self.rate = rate

    def __call__(self, length):
        self.sequence = [self.start]

        for i in range(length - 1):
            self.sequence.append(self.sequence[i] * (1-self.rate))

        print(self.sequence)
        return self.sequence

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

DM = ExponentialDecayModel(start=100, rate=0.2)
DM(length=5) # [100, 80, 64, 51.2, 40.96]
print(len(DM)) # 5
print([n for n in DM]) # [100, 80, 64, 51.2, 40.96]

print("----------------------- Task 2.6 -----------------------")
# Task 2.6
GM = ExponentialGrowthModel(start=100, rate=0.1)
GM(length=5) # [100, 110, 121, 133.1, 146.41]
```

```
GM2 = ExponentialGrowthModel(start=100, rate=0.2)
GM2(length=5) # [100, 120, 144, 172.8, 207.36]

print(GM == GM2) # 1


GM3 = ExponentialGrowthModel(start=100, rate=0.2)
GM3(length=3) # [100, 120, 144]

# print(GM == GM3) # will raise an error

print("---------------------- Task 2.7 ----------------------")
# Task 2.7
GM = ExponentialGrowthModel(start=200, rate=0.9)
GM(length=5)
GM2 = ExponentialDecayModel(start=400, rate=0.1)
GM2(length=5)
print(len(GM)) # 5
print([n for n in GM])
print(len(GM2)) # 5
print([n for n in GM2])
print(GM == GM2) # 0
print(GM == GM) # 5

GM3 = ExponentialDecayModel(start=100, rate=0.2)
GM3(length=3)
# print(GM == GM3) # will raise an error

print("---------------------- Task 3.1 ----------------------")
# Task 3.1
class CombinedBioModel(BioModel):
    def __init__(self, growth_start, growth_rate, decay_start, decay_rate):
        super().__init__([])
        self.growth = ExponentialGrowthModel(growth_start, growth_rate)
        self.decay = ExponentialDecayModel(decay_start, decay_rate)

    def __call__(self, length):
        self.growth(length=length)
        self.decay(length=length)
        self.sequence = []
        for i in range(len(self.growth)):
            self.sequence.append(self.growth.sequence[i] * self.decay.sequence[i
        print(self.sequence)
        return self.sequence
```

```
CBM = CombinedBioModel(growth_start=100, growth_rate=0.1, decay_start=1.0, decay
CBM(length=5) # [100.0, 104.50, 109.20, 114.12, 119.25]


print("---------------------- Task 3.2 ----------------------")
# Task 3.2
start1 = 10
start2 = 12
rate1 = 0.5
rate2 = 0.6
GM1 = ExponentialGrowthModel(start=start1, rate=rate1)
GM1(length=5)
DM1 = ExponentialDecayModel(start=start1, rate=rate1)
DM1(length=5)
CM1 = CombinedBioModel(growth_start=start1, growth_rate=rate1, decay_start=start
CM1(length=5)
GM2 = ExponentialGrowthModel(start=start2, rate=rate2)
GM2(length=5)
DM2 = ExponentialDecayModel(start=start2, rate=rate2)
DM2(length=5)
CM2 = CombinedBioModel(growth_start=start2, growth_rate=rate2, decay_start=start
CM2(length=5)

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

axs[0, 0].plot(GM1.sequence, color="blue", label="start = 10, rate=0.5")
axs[0, 0].plot(DM1.sequence, color="red", label="start = 10, rate=0.5")
axs[0, 0].set_title("Blue Growth, Red Decay")
axs[0, 0].set_xlabel('Time')
axs[0, 0].set_ylabel('Value')
axs[0, 0].legend()
axs[0, 0].set_ylim(-10, 150)

axs[0, 1].plot(GM2.sequence, color="blue", label="start = 12, rate=0.6")
axs[0, 1].plot(DM2.sequence, color="red", label="start = 12, rate=0.6")
axs[0, 1].set_title("Blue Growth, Red Decay")
axs[0, 1].set_xlabel('Time')
axs[0, 1].set_ylabel('Value')
axs[0, 1].legend()
axs[0, 1].set_ylim(-10, 150)

axs[1, 0].plot(CM1.sequence, color="green", label="start = 10, rate=0.5")
axs[1, 0].set_title("Combined Sequence")
axs[1, 0].set_xlabel('Time')
axs[1, 0].set_ylabel('Value')
```

```
axs[1, 0].legend()
axs[1, 0].set_ylim(-10, 150)

axs[1, 1].plot(CM2.sequence, color="green", label="start = 12, rate=0.6")
axs[1, 1].set_title("Combined Sequence")
axs[1, 1].set_xlabel('Time')
axs[1, 1].set_ylabel('Value')
axs[1, 1].legend()
axs[1, 1].set_ylim(-10, 150)

plt.suptitle("Growth, Decay, and Combined for Different Rates")
plt.legend()


plt.show()
```