Assignment_1_Jupyter_Notebooks_and_Objects_in_Python_[netid]

August 31, 2021

1 Assignment 1: Jupyter Notebooks and Objects in Python

Congratulation, if you've made it this far, you're already done the first part, get a Jupyter notebook up and running!

Hopefully you're somewhat familiar with Python syntax and numpy to follow along with what's below. If not (and even if you're even a tiny bit shakey in it, or don't already know all the nuances), I would highly recommend checking out Prof. Bagrow (via Jake VanderPlas)'s Whirlwind Tour of Python for a quick intro to syntax, as well as to your first steps with numpy and matplotlib (which you'll be using here).

We'll also be reviewing (or for some perhaps introducing) the concept of object-orientied programming in python.

For those of you not familiar with navigating Jupyter notebooks, there are many resources online, and it becomes especially hepful to become familiar with the keyboard shortcuts. Please also note that cells are executed in the order that they are run by you, not the order that they necessarily appear on the screen -- so be careful about skipping over blocks or editing previously run blocks. When in doubt, the `Kernel -> Restart & Run All` command can ensure that the cells are re-executed in the order they appear.

To get you started, some commenting and code has been included. To complete this assignment you will need to fill in any elipses (which may be a placeholder for just single line or a large block of code) with functining (and well commented) code of your own.

In [ ]:
```
# import packages

import numpy as np # numpy for arrays

import matplotlib.pyplot as plt # matplotib for plotting
plt.style.use('seaborn') # for style points
```

1.0.1 Q1: Arrays

The Fibonacci sequence is an example of simple generateive process. In the sequence, the value of a given entry is the sum of the two entiries preceeding it (e.g. `0, 1, 1, 2, 3, ...`).

Generate a (numpy) array of length 15 that contains the Fibonacci sequence, starting from the initial values 0 and 1 in the first two entries.

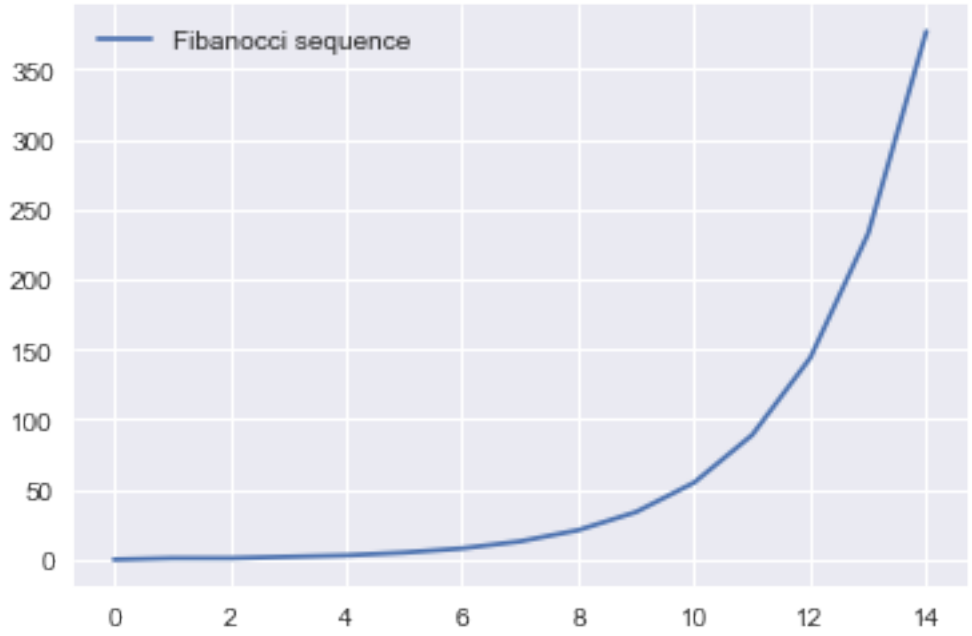
(no need to be fancy, you can just use a for-loop)

```
In [ ]: # hyperparameters
        total_iterations = 15

        # initialization
        values_over_time = [0, 1]


        # generate sequence
        for i in range(15-2):
            values_over_time.append(values_over_time[-1]+values_over_time[-2])


        # display values
        print("Fibonacci Sequence:",values_over_time)
```

1.0.2 Q2: Plotting

Now let's plot this sequence. Don't forget to always include axes labels and a legend. If you've never done this before Examples using matplotlib.pyplot.plot is a good place to start. Your plot should look something vaguely resembling this:

```
In [4]: plt.plot(values_over_time)
        plt.xlabel = "iteration"
        plt.ylabel = "value"
        plt.legend(["Fibanocci sequence"])
        plt.show()
```



1.0.3 Q3: Objects and Classes

Object-oriented programming excels at it's ability to easily maniupate the attributes of different instances of the same class. Let's review object-oriented programming (or learn about it via one of the many quick online tutorials on the topic) in python by implementing a `Finbonacci_Experiment(first_value, second_value)` class which will let you dictate the starting values of a sequence

```
In [8]: class Fibonacci_Experiment:
            def __init__(self, first_value, second_value):
                self.first_value = first_value
                self.second_value = second_value

            def run(self, total_iterations):
                values_over_time = [self.first_value, self.second_value]
                for i in range(total_iterations):
                    values_over_time.append(values_over_time[-1]+values_over_time[-2])
                return values_over_time
```

Now let's use instances of this class to plot (on the same figure) three fibonacci experiments: one with the starting two values of `(0, 1)`; one with the starting values `(1, 5)`; and one with the starting values `(7, 0)`.

```
In [9]: one = Fibonacci_Experiment(0,1)
        two = Fibonacci_Experiment(1,5)
        three = Fibonacci_Experiment(7,0)

        print(one.run(15))
        print(two.run(15))
        print(three.run(15))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
[1, 5, 6, 11, 17, 28, 45, 73, 118, 191, 309, 500, 809, 1309, 2118, 3427, 5545]
[7, 0, 7, 7, 14, 21, 35, 56, 91, 147, 238, 385, 623, 1008, 1631, 2639, 4270]
```

1.0.4 Q4: Why did we do this?

Python is a fantastic scripting language, but many people never use it for object oriented programming. So there must be some reason why it's being introduced here... Besides easily generating many replications or variatons of an experiment, how else might object oriented programming be helpful for Evolutionary Computation in particular?

(Type your response into the markdown cell below)

My initial thoughts on its usefulness in EC is for creating objects or organisms or whatever we happen to be evolving. This would allow us to have numerous objects that all have the same structure but that could evolve in a specifically defined manner that we can manipulate easily. It also allows us to update and experiment with this structure.

1.0.5 Congratulations, you made it through your first programming assignment!

Please save this file as a .ipynb, and also download it as a .pdf, uploading **both** to blackboard to complete this assignment.

For your submission, please make sure that you have renamed this file (and that the resulting pdf follows suit) to replce `[netid]` with your UVM netid. This will greatly simplify our grading pipeline, and make sure that you receive credit for your work.

Academic Integrity Attribution During this assignment I collaborated with:
Just me this time :)

In [ ]: