Computational Decision Making for Regular People

02: Python Crash Course

October 22, 2024

Nathan Davis Barrett

Today's Outline

1. What is Python / Google Colab?
2. Introduction to Google Colab
3. Key Python concepts
   - 3.1 Variables (integers, floats, strings, lists)
   - 3.2 Loops
   - 3.3 Functions
   - 3.4 Data Structures
4. Key Python packages
   - 4.1 Numpy
   - 4.2 Matplotlib
   - 4.3 Pandas (if there is time)

NOTE: Today's class might feel like a firehose. Don't worry if you don't perfectly grasp everything today. You'll have plenty of time to practice.

What is Python / Google Colab?

What is Python?

- Python is a computer language (a set grammatical rules) in conjunction with a computer program that reads written statements and performs the commands written therein.
- You can write Python code in any place you can write words (even on paper, if you want).
- But you can only execute Python code in the Python software package.

What is Google Colab?

- Google Colab is a website through which you want write and execute Python code.
- While you can (and are welcome to) install Python on your own computer, it can become difficult to ensure that all the packages are installed and configured correctly across different Windows, Mac, etc. computers.
- Google Colab handles all the installation in a (relatively) painless way.

If you do choose to install python on your own computer, I'd highly recommend the "Anaconda" installation and package management system. You can find more info here: https://www.anaconda.com/download
Please come talk to me if you'd like to do this.

Introduction to Google Colab.

- Google Colab is a nice program that has a graphical user interface that lets you see your Python code, it's output, and several other useful options all in an organized way.
- This interface is called a "Notebook".
- To launch the Google Colab notebook for today's class, please Lecture 2 folder of the course website. There, there should be an "Open in Colab" button to pull up the notebook.

Course Website:
https://github.com/NathanDavisBarrett/ComputationalDecisionMakingCourse

Go to the Lecture 02 folder and look for this button:

Python: Basic Arithmetic

$$5 \times 5 = 25$$

$$12 \times 12 = 144$$

$$123 \times 7 = ?$$

$$14 \times 17 = ?$$

```
print(123 * 7)
print(14 * 17)
```

Python: Variables

- variableName = value

Type	Description	Syntax
int	An integer number	A number with no decimal in it
float	A "floating point" (a.k.a. decimal) number	A number with a decimal in it
char	An alphabetic character	A single character in single quotes
str	A string of alphabetic characters	Several characters in double quotes
bool	A boolean True or False (binary 0 or 1) value	"True" or "False

- Exercises:
  1. Create a variable named 'x' and assign it a value of 37
  2. Create a variable named 'y' and assign it a value of 15.2
  3. Create a variable named 'z' and assign it a value of 'x + y'
  4. Print the value of 'z'

1. Create a variable named 'name1' and assign it a value of "Joe"
2. Create a variable named 'age1' and assign it a value of 37
3. Print the value of 'age1 + name1'. Before you execute the command, what do you think will happen?
4. Execute the command. What happens?

Python: Iterable Variables

Lists:

- A list of other variables / values
- listName = [__elements__]
- Access individual elements using listName[index]
- "index" is the element's position within the list
- **NOTE:** The creators of Python defined it in such a say what the "1^{st}" element in a list is located at index 0.

```
myList = ["element1","element2","element3"]
print(myList[0])
print(myList[2])
```

Sets:

- Very similar to a list but with not specific order.
- Thus there are no indices.
- setName = {__elements__}

Why the distinction?

- Sometimes the order of a collection of data is important
  - Order within a queue
  - When data is corresponding between multiple lists
  - When we want values neatly sorted
- Other times we specifically don't want things ordered
  - e.g. If I simply wanted a collection of the names of each student in class today.

Python: Functions

- For when we have a section of code that we're going to need to repeat.
- **Pro Tip:** To keep your code clean and relatively free of errors, try to minimize the amount of copying and pasting that you do.
- Syntax:

```
def functionName(parameters):
    Code to Repeat Here...
```

- Python keeps track of when a function starts and stops using indentation.

```
print((1+2)/3)
print((15+14)/5)
print((38+29)/32)
```

```
def AddThenDivide(a,b,c):
    addedNumbers = a + b
    finalResult = addedNumbers / c
    return finalResult

print(AddThenDivide(1,2,3))
print(AddThenDivide(15,14,5))
print(AddThenDivide(38,29,32))
```

Python: Loops

```
print(AddThenDivide(1,2,3))
print(AddThenDivide(15,14,5))
print(AddThenDivide(38,29,32))
```

There is still a good bit of copy and paste.
This is where we can use a loop. (**Exersize**)

```
aList = [1,2,3,4,5]
bList = [6,7,8,9,10]

aPlusBList = [0,0,0,0,0]
for i in range(len(aList)):
    aPlusBList[i] = aList[i] + bList[i]
```

```
aPlusBList = [aList[i] + bList[i] for i in
range(len(aList))]
```

- Three key players:
  - A list, set, etc. to iterate over
  - A temporary variable to hold each element individually at different times
  - Some code we want to repeat for each element
- Two different syntaxes:
  1. for tempVar in myList:
     Some Code Here
  2. [Some Code Here for tempVar in myList]
- Two related, built-in Python functions:
  - *len*(*myList*): Gives the number of elements in myList
  - *range*(*num*): Produces a list of all integers from 0 to "num"

If/Else Conditions

NOTE: The notion of an if/else condition exists in Python, but NOT in mathematical modeling. We'll cover how to re-create the behavior of an if/else condition in mathematical modeling in a couple of weeks.

Often in logic, and in code, we want to do some things only if a certain condition is or is not met. We do this in Python using a collection of *if*, *else*, and *elif* statements.

Syntax:

```
if condition1:
    Do some stuff
elif condition 2:
    Do some other stuff
else:
    Do some other other stuff
```

greater than	a > b
greater than or equal to	a >= b
less than	a < b
less than or equal to	a <= b
equal to	a == b
not equal to	a != b

```
a = 5
b = 3
if a <= b:
    print("a <= b")
elif a > b:
    print("a > b")
```

List / Function / Loop / If Exercise

Please try to complete this exercise on your own.
I'll circle around and help if anyone needs it.

Python: Custom Data Structures

A quick note on this topic: Unless you want to, you won't need create any custom data structures in this class. So it's not as important to understand exactly how this is done as much as it is important to realize that these exist. We'll be using lots of "custom" data structures that have been created by other people.

- Custom data structures are called classes.
- A class can have named variables inside of it.
- A class can also have functions inside of it.

```
people = [Person("Joe",37),Person("Sally",41),Person("Jacob",17)]

print(people[0].age)
print(people[2].name)
people[0].printInfo()
```

Python Packages

- We can import other people's code into our code!
- Syntax:
  - import packageName
  - import packageName as myPersonalAbbreviation
  - from packageNAme import specificFunctionName
- Important packages we'll use in this course:
  - numpy
  - matplotlib
  - pandas
  - pyomo

Numpy

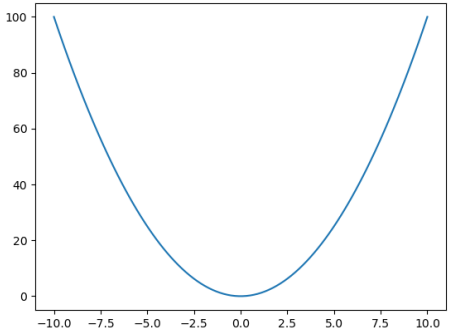
- A package for all things math
- import numpy as np
- Complex math functions like $\sqrt{}$ ("np.sqrt"), log ("np.log")
- A very simple way to deal with lists (or at least a custom version of lists called "arrays")
  - Arrays can be added, multiplied, divided, etc. automatically implementing element-wise addition, multiplication, division, etc.
  - Arrays can be automatically generated:
    - np.linspace
    - np.arange
  - Arrays can be "sliced"
- See examples in the Jupyter Notebook

Matplotlib

- A package for all things plotting or graphing
- import matplotlib.pyplot as plt
- Most important function: plt.plot()

```
xData = np.linspace(-10,10,100)
yData = xData * xData

plt.plot(xData,yData)
```



Examples: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Pandas

- Lots of uses. But in this class we'll mainly use it to import and export excel files.
- import pandas as pd
- Main Data Structure: pd.DataFrame
  - More or less a table. Each column has a specific name.
  - Columns can be accessed using "myDataFrame[columnName].to_numpy()"
  - Write to Excel: "myDataFrame.to_excel("fileName.xlsx", index=False)
  - Read from Excel: pd.read_excel("fileName.xlsx")
- See examples in the Jupyter Notebook

Next Class

PLEASE DON'T FORGET TO BRING YOUR LAPTOPS TO CLASS!

Again, Today's class might have felt like a firehose. Don't worry if you don't perfectly grasp everything today. You'll have plenty of time to practice as we move forward.