EM384: Analytical Methods for Engineering Management

Lesson 34: Control Structures in Python

24 April 2023

Table of contents

Lesson Objectives

Lesson Objectives

- Design and implement simple Monte Carlo simulations in python.
- Visualize the results of monte carlo simulation in python.
- Create and interpret an empirical cumulative distribution function (ECDF) from the results of the model.

Review

Why Python?

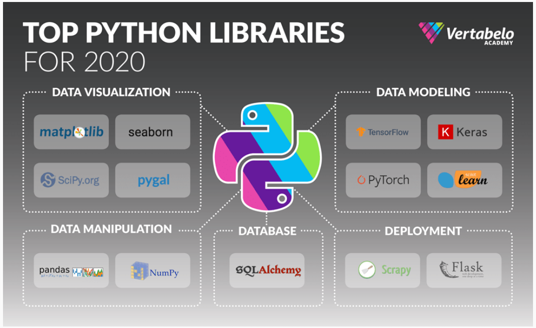
1. More powerful than Excel (especially for Monte Carlo Simulation!)
2. Code is easy to read, use and maintain
3. Compatible with major platforms and systems
4. Large standard library
5. Cost-effective approach
6. Autocompletion, Autosuggestion, Docstring
7. Other undergraduate Engineering Management programs use Python

Python is often compared to other high-level languages such as Swift, JavaScript, and C#. It is generally considered to be easier to learn and use than these languages, as it has a more concise syntax and a more intuitive object-oriented design. It is also dynamically typed, which means that you don't need to specify the data type of a variable when you declare it, which can make it easier to write code.

Libraries Explained

Python comes with the standard Library installed. This means that there are any built-in functions (e.g print(), sum(), etc) that do not require you to import other libraries. A database contains a dataset. A dataset is a table of information.

- Libraries allow you to use more advanced functions that others have already built.
- Libraries give you added functionality.
- In this class, we will use several popular libraries such as NumPy, Pandas, SciPy, and matplotlib.



Using Python Libraries

There are four ways to import a library in python (#4 is rarely used). In each case, we import the library or part of the library that we want and in this example we assign a random value between 1 and 10 to the variable ***a*** using the ***randint*** function from the ***numpy*** library.

1. Import entire library

```
import numpy
a = numpy.random.randint(0,10)
```

2. Import entire library with alias

```
import numpy as np
a = np.random.randint(0,10)
```

3. Import part of a library

```
from numpy import random
a = random.randint(0,10)
```

4. Import part of a library with alias

```
from numpy import random as rd
a = rd.randint(0,10)
```

Standard Libraries used in EM384

You can import libraries even if you don't end up using them. A standard list of Python libraries that we will use in EM384 along with their recommended import commands are given below (You instructor may add more).

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from statsmodels.distributions.empirical_distribution import ECDF
import csv
```

Commenting your Code

Commenting your Python code is good practice to make your model interpretable.

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

Using the Console as a Calculator

The console can also act as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators +, -, * and / work just like in most other languages. To raise an expression to a power, use ** (instead of ^); parentheses () can be used for grouping. For example:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a float
1.6
```

Using the Console as a Calculator

In the console, the last printed expression is assigned to the variable _. This means that when you are using Python as a desk calculator, it is somewhat easier to continue calculations, for example:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Basic Data Structures

Basic Python data structures are **integers**, **floats**, **booleans**, **strings**, and **lists**. In the variable explorer, these will appear as int, float, bool, str, and list. Python infers the type when you declare a variable

```
a = 5 #int
b = 3.14 #float
a_bool = False #boolean
my_str = 'EM384' #string.  Can also use " "
shopping = ['eggs', 'ham', 'gravy'] #a list of strings
scores = [270, 300, 245, 700] #a list of integers
mixed = ['Hello', [3,4], 2.1, True] # a list of mixed types
```

To find out the type of a variable, you can look in the variable explorer or use the type function.

```
>>> type(a_bool)
bool
```

Basic Data Structures

It is important to remember that you cannot use **reserved words** for your variable names, not can you start a variable with an underscore _ or a number.

Spyder will automatically color reserved words in a different color to let you know they are reserved words. The colors in these slides are not necessarily the same colors you will see in Spyder (depends on your theme).

E.g.

```
for and if or else while type def print del
```

Note: For these lesson slides, we use the following convention:

```
#Python code without >>> in front means I can either enter this
#command in the console or execute it in my script.

>>> #This means the command is typed in the console.
#The next line is the output
```

Python Strings

Python knows a number of compound data types, used to group together other values. One we will use is the string, used to group characters together. Strings are **immutable**, meaning you cannot change parts of strings that have been created (but you can overwrite them with new ones, and you can join strings together with concatenation).

```
greeting = 'Hello'
another = 'World'
```

And now concatenation:

```
>>> greeting + ' ' + another
'Hello World'
```

```
>>> print(greeting + ' ' + another)
Hello World
```

```
>>> print(greeting,another)
Hello World
```

Python Lists: Creating and referencing

The most versatile compound data type is the list, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type. Unlike strings, lists are **mutable**.

```
squares = [1, 4, 9, 16, 25]
```

You can also create an empty list (that you can add, or **append**, to later)

```
empty_sad = []
```

Like strings (and all other built-in sequence types), lists can be **indexed**. Remember indexing starts at 0!

```
>>> squares[3]
16
```

Python Lists: Slicing

Lists can also be **sliced** using the **:** symbol and the index/indices reference inside brackets [ ].

```
squares = [1, 4, 9, 16, 25]
```

```
>>> squares[:3]
[1, 4, 9]
```

```
>>> squares[3:]
[16, 25]
```

```
>>> squares[2:3]
[4]
```

Python Lists: Replacing values

You can **replace** an item of a list with another value using the index reference inside brackets [ ] and the assignment operator **=**.

```
squares = [1, 4, 9, 16, 25]
squares[3] = 8
```

```
>>> squares
[1, 4, 9, 8, 25]
```

Python Lists: Appending values

You can add an element to a list with another value using the **append** function. This element gets added onto the end.

```
squares = [1, 4, 9, 16, 25]
squares.append(36)
```

```
>>> squares
[1, 4, 9, 16, 25, 36]
```

Python Lists: Deleting values

You can delete an element from a list using the function and the element (or slice's) index.

```
squares = [1, 4, 9, 16, 25]
del(squares[3])
```

```
>>> squares
[1, 4, 9, 25]
```

Python Control Structures

Python Control Structures

There are three types of control structures:

- **Sequence**: Every program is controlled by the sequence of commands. The command sequence creates inherent control.
- **Iteration**: FOR loops and WHILE loops are classic iteration structures.
  Definition: an iterable is any Python object capable of returning its members one at a time, permitting it to be iterated over in a for-loop. Lists, strings, and tuples are all 'iterables'. You will commonly iterate across a numpy array or pandas dataframe.
- **Selection**: IF, ELIF, ELSE...very similar to conditional logic in Excel.

Control Structures: Selection

If statements evaluate a boolean expression and run the first block of code if true, and the second block of code if false. Each block of the **If** statements must be properly indented. Only the **if** statement and the first block is mandatory. The **elif**(s) and **else** are optional.

```
num = 4
if num > 5:
    #do this if true
    print('buggy')
```

The control sequencing for an if / elif / else statement is to EXIT the statement after executing the sub-code for the first available TRUE condition!

```
num = 4
if num < 5:
    #do this if true
    print('buggy')
#check this if false
elif num == 4:
    #do this if true
    print('very buggy')
else:
    #do this if false
    print('correct')
```

Control Structures: Iteration

For loops run a block of code using an **iterator**. The loop always runs for a predetermined number of iterations until the iterator reaches the maximum value.

```
scores = [3,6,4,8,9,2]
for i in range(0,5):
    print('the score is',scores[i])
```

When using a FOR loop with i in range(a,b), remember that the loop EXITS when it sets i=b, thus it will NOT execute the loop for i==b.

```
meals = ['bfast','lunch','dinner']
for i in meals:
    print('the meal is',i)
```

Is there any difference in output between the code above and below?

```
meals = ['bfast','lunch','dinner']
for i in range(0,4):
    print('the meal is',meals[i])
```

Control Structures: Iteration

While loops run a block of code using a **boolean expression**. Each time the loops starts again, it checks the boolean expression and only exits the loop if it evaluates to **False**.

```
counter = 0
while counter <= 10 :
    print('the counter is at',counter)
    counter = counter + 1
```

while loops can result in an infinite loop if not careful. To exit an infinite loop and stop the 'kernel', click the red square at the top of the console window.

Coding Questions

Question 1

Which of the following lines of code results in an error? why?

1.
   ```
   _my_var = 10
   ```
2.
   ```
   my2_var = 10
   ```
3.
   ```
   for = 10
   ```
4.
   ```
   2_var = 10
   ```
5.
   ```
   2'_'var = 10
   ```

Question 1 - Answer

Which of the following lines of code results in an error? why?

1.
```
_my_var = 10        #cannot starts with a symbol character
```
2.
```
my2_var = 10        #good!
```
3.
```
for = 10            #cannot be a reserved word
```
4.
```
2_var = 10          #cannot start with a number
```
5.
```
2'_'var = 10        #cannot have quotation marks in it
```

Question 2

What would be the output in the console for the following Python script?

```
print(hello)
```

Question 2 - Answer

What would be the output in the console for the following Python script?

```
print(hello)
```

*NameError: **name 'hello' is not defined***. We would get an error because hello is an undefined variable ('hello' is a string and would work)

Question 3

What would be the output in the console for the following Python script?

```python
for i in range(0,5):
    print(i)
```

Question 3 - Answer

What would be the output in the console for the following Python script?

```
for i in range(0,5):
    print(i)
```

```
0
1
2
3
4
```

Question 4

What would be the output in the console for the following Python script?

```python
some_list = [10,20,30,40]
del(some_list[1])
some_list.append(15)
print(some_list)
```

Question 4 - Answer

What would be the output in the console for the following Python script?

```python
some_list = [10,20,30,40]
del(some_list[1])
some_list.append(15)
print(some_list)
```

[10, 30, 40, 15]

Question 5

What would be the output in the console for the following Python script?

```
a = 7
b = a + 2
c = [a,b]
c.append(a)
print(c)
```

Question 5 - Answer

What would be the output in the console for the following Python script?

```
a = 7
b = a + 2
c = [a,b]
c.append(a)
print(c)
```

```
[7, 9, 7]
```

Question 6

What would be the output in the console for the following Python script?

```
course_list = [3,6,4,7]
for i in range(0,len(course_list)):
    print(i)
```

Question 6 - Answer

What would be the output in the console for the following Python script?

```python
course_list = [3,6,4,7]
for i in range(0,len(course_list)):
    print(i)
```

```
0
1
2
3
```

Question 7

What would be the output in the console for the following Python script?

```
course_list = [3,6,4,7]
for i in course_list:
    print(i)
```

Question 7 - Answer

What would be the output in the console for the following Python script?

```
course_list = [3,6,4,7]
for i in course_list:
    print(i)
```

```
3
6
4
7
```

Question 8

What would be the output in the console for the following Python script?

```
course_list = [3,6,4,7]
for i in range(0,4):
    print(course_list[i])
```

Question 8 - Answer

What would be the output in the console for the following Python script?

```python
course_list = [3,6,4,7]
for i in range(0,4):
    print(course_list[i])
```

```
3
6
4
7
```

Question 9

What is the value of **stonks** and **squeeze** after running the following Python script?

```python
stonks = 10
moon = 99
squeeze = False
if stonks > moon:
    squeeze = True
    Stonks = 1000
else:
    stonks = 0
```

Question 9 - Answer

What is the value of **stonks** and **squeeze** after running the following Python script?

```python
stonks = 10
moon = 99
squeeze = False
if stonks > moon:
    squeeze = True
    Stonks = 1000
else:
    stonks = 0
```

```
>>> stonks
0
>>> squeeze
False
```

Question 10

What is the value of **stonks** and **squeeze** after running the following Python script?

```python
stonks = 10
moon = 99
squeeze = False
while squeeze == True:
    stonks = stonks + 1
    if stonks > moon:
        stonks = 1000
```

Question 10 - Answer

What is the value of **stonks** and **squeeze** after running the following Python script?

```python
stonks = 10
moon = 99
squeeze = False
while squeeze == True:
    stonks = stonks + 1
    if stonks > moon:
        stonks = 1000
```

```
>>> stonks
10
>>> squeeze
False
```

Question 11

What is the value of **stonks** and **squeeze** after running the following Python script?

```python
stonks = 10
moon = 99
squeeze = False
while squeeze == False:
    stonks = stonks + 1
    if stonks > moon:
        squeeze = True
```

Question 11 - Answer

What is the value of **stonks** and **squeeze** after running the following Python script?

```python
stonks = 10
moon = 99
squeeze = False
while squeeze == False:
    stonks = stonks + 1
    if stonks > moon:
        squeeze = True
```

```
>>> stonks
100
>>> squeeze
True
```

Practical Exercise

PE1 Military Pay

Create a new Python file and import the military pay table by running the following code:

```
import pandas as pd
import numpy as np
url_csv = 'https://raw.githubusercontent.com/evangelistapaul/MC_EM384/main/Sim_lab_military_pay.csv'
df_pay = pd.read_csv(url_csv)
```

1. Iterate through every month (row) of the data.
2. If you receive a pay raise, print the following statement:
   MONTHLY PAY RAISE!! Previous pay: <previous pay>, New monthly pay: <new monthly pay>, Pay raise amount: <amount of raise>
3. If you are promoted, print the following statement:
   PROMOTION: You are now a <new grade>
4. **BONUS:** in addition to printing your new grade, print your new rank.

Conclusion

Next Class

Homework:

- Review Block 4 readings and previous Python tutorial videos.

Next Lesson:

- Design and implement simple Monte Carlo simulations in python.
- Visualize the results of monte carlo simulation in python.
- Create and interpret an empirical cumulative distribution function (ECDF) from the results of the model.