Crash Course in Python

Niharika Sravan and GionMatthias Schelbert

Purpose

Since 2003, Python has ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of January 2016, it is in position five. Large organizations such as Google, Yahoo!, and Dropbox use Python for development.

There are good reasons why Python is rivaling conventional programming languages such as C++ and Java in popularity. First, Python emphasizes code readability, often using English keywords where other languages use punctuation. It is designed to have an uncluttered visual layout, using whitespace indentation, rather than curly braces or keywords, to delimit blocks. Second, its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. Third, Python has a large and comprehensive standard library providing tools suited to many tasks.

While there are innumerable resources available to learn Python, there are none that allow an experienced programmer to jump into coding with minimal time and effort. This lesson aims to satisfy this need by having the student focus on small tasks designed to introduce them to Python syntax and capabilities.

Overview

This lesson serves as a launching pad to coding in Python for students who have experience coding in other languages. It will introduce them to Python syntax and programming essentials by using different approaches to accomplish a single task.

Student Outcomes

After completing this lesson, students will be able to

- develop Python code for any purpose
- create sophisticated data visualization

Standards Addressed

Performance expectation: DCI codes

HS-ETS1-2. Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.

HS-ETS1-3. Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics as well as possible social, cultural, and environmental impacts.

Reach for the Stars is a GK-12 program supported by the National Science Foundation under grant DGE-0948017. However, any opinions, findings, conclusions, and/or recommendations are those of the investigators and do not necessarily reflect the views of the Foundation.

HS-ETS1-4. Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Time
3 class periods or approx. 3 hours

Level
11th - 12th grade

Materials and Tools
- ✓ Access to a computer that has a
  - o a text editor,
  - o a Python interpreter, and
  - o (preferably) access to the internet.
- ✓ A printed copy of the handout (appended at the end of this lesson outline)

Preparation
Ensure that students have a printed copy of the handout (5 pages).

Prerequisites
Students should have experience coding in any other language.

Background
None

Teaching Notes
Since this lesson is intended for students with programming experience, they will primarily be working through the handout independently. The teacher should help with troubleshooting technical issues and answering student questions. The teacher should encourage students to troubleshoot on their own (using online resources) and develop computational thinking.

One way to administer the lesson would be to have students work through the handout one exercise at a time. First, ask the students to type out the code (in `courier font`) and run it once (with or without errors). Then walk through the code line by line, discussing the algorithm and syntax. Then ask the students to work on their codes until they run correctly.

Teachers should work through the exercises themselves before assigning it to students. This will allow them to anticipate the problems students might have, saving class time otherwise spent on troubleshooting.

Assessment
Students should be execute each code in the handout correctly.

Additional Information
An excellent interactive and comprehensive resource for learning Python (and many other programming languages) is https://www.codecademy.com/.

The handout starts on the next page.

Learning Python

Python is a high-level programming language. It has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks.

Today, we will learn Python by focusing on writing scripts to calculate the average score of a class of students in different ways. The exercises are meant to introduce you to the Python syntax and capabilities.

Ex 1. Calculate the average score of five students. Do this by first defining five variables, say: `score_1`, `score_2`, `score_3`, `score_4` and `score_5`. Initialize them with whatever values you like. Store their sum in a variable called `sum`. Then, divide sum by `5` and store the result in `average`. Finally, print `average`.

Here is what your script might look like:

```
score_1 = 78.0
score_2 = 81.5
score_3 = 98.
score_4 = 69
score_5 = 92e0

sum = score_1 + score_2 + score_3 + score_4 + score_5
average = sum/5.

print 'The average is', average
```

Note that python does not need to know if the scores are of integer or floating type!

You can run the script by typing `python <scriptname>` (Your script name should end with `.py`).

The output of the above code would be:

```
The average is 83.7
```

Ex 2. Now ask the user to enter the scores. They should be allowed to enter as many scores as they like. You should compute the average of the scores entered and display it.

This can be achieved in many ways but we will implement it using:

a. the `for` loop, and
b. the `while` loop.

Here's the idea: First, we will print a welcome message and ask the user if they know how many scores they want to enter. If they say yes, we will start a `for` loop and sequentially collect all the scores. If they say no, we will start a `while` loop and collect the scores iteratively.

Here is what your script might look like:

```
response = raw_input('Do you know how many scores you have for
computing average? (y/n) ')

if response == 'y' or response == 'yes':
      num = int(raw_input('How many scores do you want to enter? '))
      if num <= 0:
                  print 'Number of scores cannot be <= 0. Quitting!!'
                  exit()
      #use for loop
      sum = 0.
      for i in range(num):
                  score = float(raw_input('Enter score #'+str(i+1)+'
'))
                  sum += score
      avg = sum/num
      print 'The average score is', avg
elif response == 'n' or response == 'no':
      #use while loop
      i = 0
      sum = 0.
      response = 'y'
      while response == 'y' or response == 'yes':
                  score = float(raw_input('Enter score #'+str(i+1)+'
'))
                  i += 1
                  sum += score
                  response = raw_input('Do you want to enter more
scores? (y/n) ')
      avg = sum/i
      print 'The average score is', avg
else:
      print 'Your response was invalid. Quitting!!'
```

The code in blue is executed when the user knows how many scores they have. The user is asked for the number of scores they wish to enter. A `for` loop is then used to receive the scores and the average displayed.

The code in green is executed when the user does not know how many scores they have. The user is asked after each score is entered whether they want to enter more scores. If they do not say `y/yes` anymore, the average is displayed.
Finally, the code in red is executed when the user gives an invalid response.

Here are a couple of example executions of the above code:

```
Do you know how many scores you have for computing average? (y/n) yes
How many scores do you want to enter? 3
Enter score #1 78
Enter score #2 91
Enter score #3 88
The average score is 85.6666666667
```

and

```
Do you know how many scores you have for computing average? (y/n) n
Enter score #1 78
Do you want to enter more scores? (y/n) yes
Enter score #2 91
Do you want to enter more scores? (y/n) y
Enter score #3 88
Do you want to enter more scores? (y/n) meh
The average score is 85.6666666667
```

Ex 3. Finally, we will compute the average by reading the scores from a file. In this example, we will be working with a file called ‘scores.dat’ containing the scores of 10 students in 5 subjects. The contents of this file are:

Annu	97	95	91	100	88
Diya	56	81	76	72	56
Geet	63	74	83	84	77
Hari	78	69	89	84	75
Lata	59	63	93	79	82
Neha	85	92	99	74	63
Prit	45	56	51	75	66
Ravi	67	78	54	90	73
Raju	86	93	71	75	90
Tanu	80	87	94	98	73

To do this, we will use a couple of Python’s in-built functions belonging the class `pylab`. The first will read an entire column from the file into an array. The second will compute the average of the elements in it.

Here is what your script might look like:

```
import pylab as py

file = 'scores.dat'
col = 1
scores = py.loadtxt(file,usecols=(col,),dtype='int')
print scores
avg = py.average(scores)
print 'The average score is', avg
```

The output of the above code would be:

```
[97 56 63 78 59 85 45 67 86 80]
The average score is 71.6
```

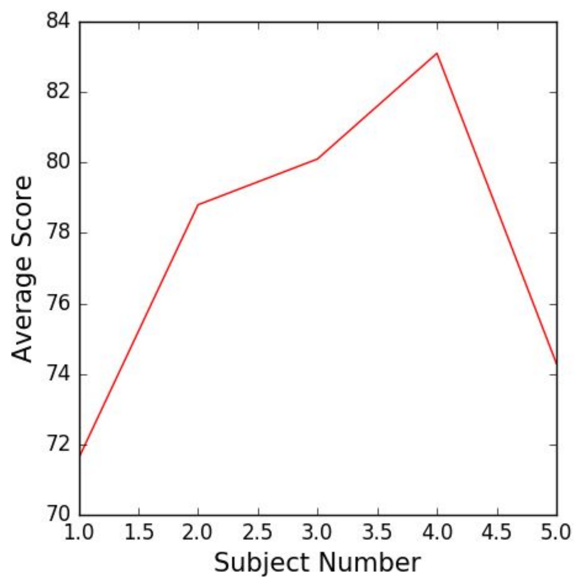
Now, let us try and visualize the data.

The following code will generate line and scatter plots of the average of the 5 subjects:

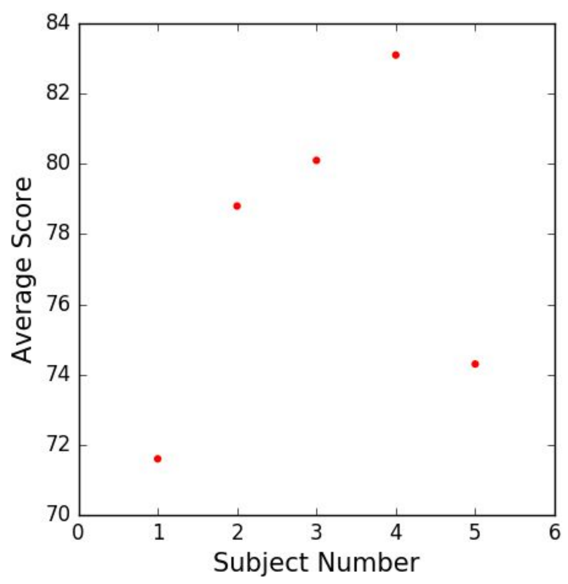
```
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
import pylab as py

file = 'scores.dat'
avg_scores = py.zeros(5)
sub_num = py.arange(1,6)
for i in range(5):
      col = i+1
      scores = py.loadtxt(file,usecols=(col,))
      avg_scores[i] = py.average(scores)

fig = plt.figure(figsize=(5,5))
#plt.plot(sub_num, avg_scores, color='red')
plt.scatter(sub_num, avg_scores, facecolors='red', edgecolors='none')
plt.xlabel('Subject Number', fontsize=15)
plt.ylabel('Average Score', fontsize=15)
fig.savefig('avgs.png')
plt.show()
```

Note: the plot files are specified here as png files but you can choose pretty much any format you like.

Here are what these line and scatter plots look like:



line



scatter

You can format your plots in numerous ways to give it the look you desire. You can choose your own colors, line styles, line weights, scatter point shapes, font styles etc.

Finishing thoughts:

The great thing about Python is the wealth of online resources. If you are ever curious about how to accomplish something or have an error you need help troubleshooting, you should first Google the question + ‘Python’ and more often than not you’ll find that somebody will already have asked and answered your question, usually on stackoverflow.