Reading Python Code (1/18)

Learning Objectives:

- Translate a given task into steps that a computer can follow. (Code)
- Describe a set of basic Python syntax and commands. (Code)
- Predict the output of simple Python code. (Code)

Important Concepts

1) How a computer "thinks," so that we can give it instructions it can understand. Concepts covered includes: **algorithm, command, syntax, CPU, memory, input/output, pseudocode**.

2) Basic elements of the Python language: **objects and variables, conditional statements, for loops, functions and packages, and input/output.**

3) Reading a simple piece of python code to predict what it will do.

Please watch the lecture video for more details.
The code example we had in class was:

```
[3]: l=[7,2,5,4]
     for i in range(len(l)):
         for j in range(i+1,len(l)):
             if l[i]>l[j]:
                 tmp=l[i]
                 l[i]=l[j]
                 l[j]=tmp
     print(l)

[2, 4, 5, 7]
```

To walk through the example, we will make a print statement at each loop so we see the inner working of the code. This is a useful strategy to understand a new piece of code. Note that I'm using the command string.format(). See documentation here: https://www.digitalocean.com/community/tutorials/how-to-use-string-formatters-in-python-3

```
[17]: l=[7,2,5,4]
      print ('Input list l={0}'.format(l))
      for i in range(len(l)):
          print('Executing outer for loop with i={0}'.format(i))
          for j in range(i+1,len(l)):
              print ('\tExecuting inner for loop with j={0}'.format(j))
              print ('\t\tComparing l[{0}]={2} and l[{1}]={3}: '.format(i,j,l[i],l[j]),end=""
              if l[i]>l[j]:
                  print ('{0} is bigger than {1}, swap:'.format(l[i],l[j]))
                  print ('\t\t list before swap: l={0}'.format(l))
                  tmp=l[i]
                  l[i]=l[j]
                  l[j]=tmp
```

```
                print ('\t\t list after swap: l={0}'.format(l))
            else:
                print ('{0} is not bigger than {1}, do nothing'.format(l[i],l[j]))
        print()
    print('Final output:')
    print(l)
```

```
Input list l=[7, 2, 5, 4]
Executing outer for loop with i=0
        Executing inner for loop with j=1
                Comparing l[0]=7 and l[1]=2: 7 is bigger than 2, swap:
                 list before swap: l=[7, 2, 5, 4]
                 list after swap: l=[2, 7, 5, 4]
        Executing inner for loop with j=2
                Comparing l[0]=2 and l[2]=5: 2 is not bigger than 5, do nothing
        Executing inner for loop with j=3
                Comparing l[0]=2 and l[3]=4: 2 is not bigger than 4, do nothing

Executing outer for loop with i=1
        Executing inner for loop with j=2
                Comparing l[1]=7 and l[2]=5: 7 is bigger than 5, swap:
                 list before swap: l=[2, 7, 5, 4]
                 list after swap: l=[2, 5, 7, 4]
        Executing inner for loop with j=3
                Comparing l[1]=5 and l[3]=4: 5 is bigger than 4, swap:
                 list before swap: l=[2, 5, 7, 4]
                 list after swap: l=[2, 4, 7, 5]

Executing outer for loop with i=2
        Executing inner for loop with j=3
                Comparing l[2]=7 and l[3]=5: 7 is bigger than 5, swap:
                 list before swap: l=[2, 4, 7, 5]
                 list after swap: l=[2, 4, 5, 7]

Executing outer for loop with i=3

Final output:
[2, 4, 5, 7]
```

FAQ:

Q: In the example above, why do we not use `range(len(l)-1)`, since python counts from 0?

A: `range(4)=[0,1,2,3]` already, so this is taken into account by how the `range` is defined in Python. If we had used `range(len(l)-1)` or `range(3)`, we would only get `[0,1,2]`. The fact that `range(a,b)=[a,a+1,...,b-1]` rather than up to `b` is a confusing part of the language, but as with any language there are quirks one must get used to.

Q: What is the meaning of "pseudo-code"?

A: The word pseudo means "fake," so pseudocode is not really correct code in terms of commands that computer can run, but an algorithm written using *computer logic but human words*. However, the hardest part of programming is generally translating the task into "computer

thinking," and from pseudocode it is simply a matter of Googling the correct syntax to turn it into real code.

Q: When should we use Jupyter notebook versus Spyder?

A: Jupyter notebook is ideal if we would like to share the result with another. You can simply send the .ipynb file and the other can read it. It also allows writing cells in "markdown," which is not Python code but allows easy text formating. (All the course notes and homeworks are written using Jupyter notebooks.) The drawback is that it cannot edit a .py file, so we have to put all of the code into the boxes. Spyder is for developing a more serious application, with potentially many files. It has multiple windows, one console for running (like Jupyter notebook), and one for editing files.

References

There are many freely available Python resources on the Internet. In class we covered the basic logic and syntax of Python programming, which opens the door for you to learn from more advanced resources. As with any language, learning requires exposure, so I encourage you to check out these resources.

1. Simple step by step tutorial of all Python commands we covered today, with online practice: https://www.learnpython.org/
2. Basics of using Jupyter notebooks: http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Notebook%20Basics.html
3. Seven minute video introduction to Jupyter notebook: https://www.youtube.com/watch?v=jZ952vChhuI
4. Two minute video introduction to Spyder: https://www.youtube.com/watch?v=8JiWEZEnJ40