Skill-01-IntroPython-Part1-Student

January 9, 2019

The Jupyter/IPython Notebook

This is a *jupyter* notebook. Jupyter is a programming environment that runs in a web browser. It combines documentation *cells* with code cells. What you are now reading is a documentation cell. If you want to see what the input to a documentation cell looks like, double-click this text. To switch back to the formatted documentation press `<shift><enter>`. Documentation cells can also typeset mathematical equations in line like $r = \pi$, or on their own lines like

$$e^{i\theta} = \cos\theta + i\sin\theta$$

The *code cells* look something like

```
In[3]:  print("Hello")
```

you will see plenty of them below. One nice feature is that you can edit and re-run a cell many times in Jupyter.

IPython is one of the programming languages that Jupyter supports. IPython stands for *Interactive Python*. It is a shell that runs python programming code in a very convenient interactive fashion.

1 Introduction to Python

Python is a general programming language that is widely used in scientific computing. It is open source, free, and available on many different types of computers including supercomputers, Windows and Mac personal computers, to tiny single chip micro processors.

Before we get started, please note: this notebook will not teach you basic programming. Instead, we assume you already know at least a little bit about: * variables, * loops, * conditionals (i.e., `if` and `else`), * lists or arrays, and * how to read data from a file.

What this notebook will do is show you how to do these things in Python. More specifically, how to do them well in Python.

2 Variables

A name that is used to denote something or a value is called a *variable*. Python is not a strongly typed language and variables are not declared before being used. The following examples create four variables, an integer, two floats, and a string.

```
nPoints = 23
volts1 = 1.34
amps2 = 2.34e6
msg = "Hello, world!"
```

Printing is easy in python. The simplest form is the `print` function with the items to print as function arguments Type the first four lines above lines in the cell below, then add the lines

```
print(nPoints)
print(volts1, amps2, msg)
```

(Note: Jupyter helps you out sometimes by adding an ending " when you type the opening " to start a string.) To execute the cell, press `<shift><enter>`.

You should also note that I strongly encourage the use of longer variable names that are descriptive. Years of experience have taught me that it is very hard to understand or debug code where all the variables have names like `x` and `i`. (The one exception I make is if I have a simple loop I will use a single letter for the loop variable.)

3 Go Ahead!

Cut and paste the six code lines in the cell below and run the cell. The results should look like this:

```
23
1.34 2340000.0 Hello, world!
```

```
In [ ]:
```

4 Lists

The next data topic is *lists*. A *list* is an ordered collection of objects. Again, python is liberal in allowing you to create lists on the fly and to mix data types in a list. A list is created by enclosing data objects with square brackets. The following are examples of lists in python:

```
names = ["Michael", "Vickie", "Linda", "John"]
ages = [18, 16, 12, 15]
balances = [132.28, 254.03, 76.30, 143.3]
info = ["John", 15, 143.3, names]
```

Go ahead and create the four lists of different types in the cell below. Make each list have at least four items. At the bottom of the cell print each of them out. (I think you can guess how to do that.)

Oh, and in the example the list `names` is an item in the list `info`, so a list contains a list!

4.1 Cool Jupyter Hint - Tab Completion

Jupyter keeps track of what variable names (and other names) are in your programming environment. This allows you to save time and possible spelling mistakes by typing the beginning of a name, then pressing the `<tab>` key and the name will automagically complete. If the name is not unique, a list of possibilities pops up. You can either keep typing or use the up and down arrow keys, then press the `<enter>` key to pick the right name. After you type in your lists, try typing `pri<tab>` and `print` will complete for you. Then try typing the first couple of letters of your variable name and tab completing that, too.

```
In [ ]:
```

5 Indexing and Slicing a List

Once you have a list, you can access its members in different ways by *indexing* and *slicing*. Indexing a list is just picking one item out. Python, like most other programming languages, starts indexing with `0` and the index of the last item in a list of `N` items is number `N-1`. The syntax of indexing is the name of the list with the index in square brackets, so the second item of the first list above is `names[1]`. You can print an item, use it in a math calculation, or anything you could do with a single values variable.

Go ahead and print the first item of the list `names`, the second item of the second list, etc. in the cell below.

```
In [ ]:
```

5.1 More Indexing

Python also has a feature that a **negative** index counts backward from the end of the list, so item `-1` is the last item in the list, `-2` is the second to last item, etc. Different from most languages, but clever.

5.2 Slicing a List

A *slice* of a list is a way of pulling out a sublist. The syntax for this is `list[start:end+1]` where `start` is the first item in the sublist, and `end` is the last. Note the second number in the slice is the ending index plus one. You can even use slices with negative indexing mentioned above.

Slicing can also used implied start and end indices by leaving out one of the indexes in the slice. Try printing out `balances[:3]`, `names[2:]`, and `ages[:]`. Do you understand what happens?

In the cell below, print out the next to last item from the list `names` using a negative index, and the middle two items of the `ages` list using a slice, and the implied start and end example from above.

```
In [ ]:
```

5.3 Other List Operations

There are many operations you can do with a list; I will just mention one more: appending. A common task is to build a new list based on existing lists. ## Lists are Objects At heart python is an object-oriented language. This means items have functions attached to them. These functions are called *methods* or *member functions* in other programming languages. The syntax for calling a method is `object.method()`. You can append an item ot a list using the `append` method. The usage is

```
list.append(item)
```

In the cell below add `"Anita"` to the list `names`, `15` to `ages`, `325.01` to `balances`, and `32` to `info`, then print out each modified list.

```
In [ ]:
```

5.4 Making Consecutive List of Numbers

The `range(start,end,skip)` function returns an object that *iterates* over integers. If you want a list of number, you can type `list(range(start, end, skip))`.

You can leave out some of the arguments, try printing `list(range(5))`, `list(range(2,7))`, and `list(range(0,10,2))` to figure out how it works.

Experiment below, then your final code line should print out the numbers 3, 6, 9, 12, 15.

```
In [ ]:
```

6 Definite Loops

Definite loops are a section of commands that are repeated a known number of times. In python these work differently than you are probably used to. Loops use a `for` syntax, however instead of having a start condition, a stop condition, and an incrementing condition, python loops execute over an *iterable*, that is an object that returns every item in a list. For example, you can print a greeting for each name in the list `names`. This code would do that:

```
for name in names:
    print("Hello, ", name)
```

The new variable `name` takes on the next value in the list for each iteration.

Note the body of the `for` loop is indented by one tab space. Python does *not* use curly brackets to mark the beginning and ending of the body of a `for` loop, it uses the indentation! So your indentation is very important and it has to be consistent.

Go ahead and try this example of a `for` loop in the cell below.

```
In [ ]:
```

6.1 More on Looping

Many times you have multiple lists of the same length, and you want to loop simultaneously over all the lists.

You need a integer variable that runs from the first item to the last item. OK. The `range` function will make a list of sequential numbers, but how do I know how long the lists are?

The function `len()` will return the length of a list. So here is the above `for` loop coded using and index:

```
for i in range(len(names)):
    print("Hello", names[i])
```

Go ahead and try this `for` loop.

In [ ]:

Now that we can loop using an index, we can use the same index for each list. Modify the above loop so at each iteration it uses all of the lists. Try printing out this equivalent line for each name:

```
Michael is 18, has a balance of 132.28, and info is John
```

In [ ]:

7 Functions

You need to know how to create and use *functions* in python. A lot of important operations, like reading from files, plotting, and math, are done by functions. To create a function the syntax is:

```
def myFunction(arg1, arg2):
    return arg1 + arg2**2
```

This function returns $arg1 + (arg2)^2$. Notice that the power operator in python is `**`. Calling a function is easy, just type the name of the function a `(`, the arguments separated by commas, and a closing `)`. For example

```
print(myFunction(2, 3))
```

will print the number `11`.

Type the function and print out the function value for the arguments `3` and `2` in the cell below.

In [ ]:

8 Problem

(This might be a challenge for you.)

Functions can return any type of object in python. Write a function in the cell below that has as arguments two lists that you assume are numbers. The function should loop over the lists and calculate `item1 + item2**2` for each element, then return a list of these values.

Remember the `append` method for lists? You should use that to build your list of answers. *Hint*: near the top of your function a line

```
answers = []
```

will create an empty list. Then each time through the loop append the new result. After the function is defined, call with lists of numbers for each argument to test that it is working, for example print `myFunction([1,2,4], [5,6,8])`.

Hint: The answer is `[26, 38, 68]`.

Don't give up until you solve this!

```
In [ ]:
```

9 Conclusion

This ends the first skills homework for python. This tour through variables, lists and function should get you started. Feel free to contact me with any questions by e-mail.

The next topics are libraries, arrays from the `numpy` library, and reading and writing data files.

BTW, the line below does some formatting for the Jupyter notebook. You should be able to see that it defines a function, then calls it.

```
In [12]: from IPython.core.display import HTML
         def css_styling():
             styles = open("custom.css", "r").read()
             return HTML(styles)
         css_styling()

Out[12]: <IPython.core.display.HTML object>
```