Lecture 4

Steven Walton
PS 232 Computational Methods
Department of Physics
Embry-Riddle Aeronautical University
Prescott, AZ 86301

October 30, 2014

Modules

Now that we know how to make our own functions we may want to make our own modules and header files. Like when we import Numpy for a function we may want to import a custom module that contains functions that we have made. One thing to actually note is that every python program IS a python module. But I'm not going to just stop here an say good job, you already did it. We do need to know at what times we have access to our modules. There are only "two" locations that we can place it that we can run it. It needs to either be located in the current working directory, the same as the python program you are working on, and within the directories that are listed in sys.path. I have included, in our function folder, a program that will output your system path for you. But I am also including the source code here, so you can see it.

```
import sys
from pprint import pprint as p

def syspath():
    return p(sys.path)

syspath()
```

The reason we use pprint (pretty print) is because it shows the directories in a much nicer format than with just print. Try print yourself to see the difference. If your modules are loaded in these paths then python will be able to find it.

Variadic Functions

Sometimes we want to create functions that don't always have to take the same number of inputs. If you are familiar with C/C++ coding this will not be too difficult to understand. We will be using two new symbols, '**'. We are use to writing a function with a specific number of inputs, like the following.

```
def foo(a,b,c):
   print [a,b,c]
```

If we run foo, we will get back the parameters we put in for a,b, and c. If we give more or less than three arguments then foo won't run and we will get an error. Now let's try the same type of function but with our new '*' character. (If you know C/C++, do not confuse this with a dereferencing operator).

```
def foo(*args):    # You do not have to name it args. But it is common to
                   # see this. Just like it is common to see foo as a
                   # made up function.
    return args
```

Now if we run foo we will get back the arguments, no matter how many you put in. You will notice something funny though, if you only put in one argument you will get a trailing comma at the end. This is because the return is a tuple and thus we have a trailing comma. But there are some fun things that we can do with all of this. One example is that we need a minimum number of arguments. So let's make a return statement of this.

```
    def foo(*args):
        if len(args)<2:
            return ''ERROR!!! There must be at least two arguments to foo.''
        else:
            return args
```

This will return the error statement if we do not have two or more arguments to the function. As you are getting more advanced with programming there is something that I want you to keep in mind. It won't always be you that is using your code. And you should never assume that the user knows how to use the code or what to do with it. This is why I have stressed to comment well before. But more importantly than that is to predict this in your code. If our user only puts in one argument we don't want them to just get back a empty screen, we want them to know that they need at least two arguments or else they are going to get extremely frustrated and you will not hear the end of it. It will also be important for security and speed, but we won't be covering this. Just know that you may want to not accept all commands that the user passes to the arguments. Besides, a friendly error message is going to save them time and frustration. There is one other thing I want to note before we move to keyword arguments. It is that we can pass a predefined variable back into our functions. For example if we have a variable $a = [1, 2, 3]$ and we execute $foo(*a)$ we will be returned the numbers $(1, 2, 3)$
We also mentioned the '**' operator. But first I have to introduce you to something, dictionaries. No, not the dictionary that is sitting around your house (though you can always increase your vocabulary). If you remember arrays, I'd be concerned if you didn't, you will recall that we can call the first element of array a by $a[0]$. This is nice and all, but maybe we want to name things logically and be able to call them differently and not remember their order at all. We can actually do this quite easily. First thing to note, though, is that 0 is a key for a, and that it relates it to a certain value. So let's make our own dictionary.

```
MontyPython={'John':123, 'Cleese':456, 'Skit':789, 'Spam':012}
```

You will notice if we just output $MontyPython$ that we get what we put in. But if we run $MontyPython['Cleese']$ we will get back 456. We can also run $MontyPython.keys()$ and we will get back the key words that we used. We can easily add a new key to our function, $MontyPython['Lumberjack'] = 9024$, or delete one by $del\ MontyPython['Spam']$ Which of course deletes or adds the associated value. Note that dictionaries are incredibly fast, and so it is nice to look up words and values with this method. We can do many things like verify if a key is in our dictionary (using the in function), get length of dictionary, we can also get the associated value by get $(printMontyPython.get('Lumberjack'))$, loops and all kinds of other things.

I highly suggest playing around with dictionaries, especially before moving onto the next section.

Now that we understand the keywords we can understand the $**kwargs$ that we mentioned earlier. You may recognize that this now is a shortanded version of "Key Word Arguments". So let's make a table of the skits in Monty Python Flying Circus episodes.

```
import pprint as p
def InEpisode(titlestring, **kwargs):
  p.pprint(titlestring)
  for count, skit in enumerate(kwargs):
     p.pprint('Skit {0} - {1}'.format(count,skit))
```

Now we will want to make a list of the skits in the episode.

```
 episode = ['Whither Canada?']
 skits = {``It's Wolfgang Amadeus Mozart'':1, ``Famous deaths'':2,
          ``Italian Lesson'':3, ``Whizzo Butter'':4,
          ``'It's the Arts''':5, ``Arthur 'Two Sheds' Jackson'':6,
          ``Picasso/Cycling Race'':7,
          ``The Funniest Joke in the World'':8}
# Notice that I fit to the line.  Good coding style
 InEpisode(*episode,**skits)   # Pointing to episode and skits
```

You will now notice that we got a full listing of the episode (it doesn't really look different when using pprint). Now if you had a full listing of skits for each episode we would easily be able to output episodes with their skit listings. This may seem like a difficult task, but it is actually easy because we can pull these lists from wikipedia and use the JSON data (I suggest you look this up if you are interested in pulling stuff from webpages.) There are many things that we can do with these types of functions and they have a lot of different uses. Try to be creative, programming is all about problem solving. All I will do is give you some tools to help you along your way, but the creativity is left for you.