Previous: (3a) Installing Python at Home

Next: (4a) Installing PuLP at Home

(3b) Basic Python Coding

In this course you will learn basic programming in Python, but there is also excellent Python Language reference material available on the internet freely. You can download the book "Dive Into Python" or there are a host of Beginners Guides to Python on the Python Website. Follow the links to either "BeginnersGuide/NonProgrammers" or "BeginnersGuide/Programmers" depending on your level of current programming knowledge. The code sections below assume a knowledge of fundamental programming principles and mainly focus on Syntax specific to programming in Python.
Note: >>> represents a Python command line prompt.

Commenting in Python

Commenting at the top of a file is done using """ to start and to end the comment section. Commenting done throughout the code is done using the hash # symbol at the start of the line.

Lists

A list is simply a sequence of variables grouped together. The *range* function is often used to create lists of integers, with the general format of range(start,stop,step). The default for start is 0 and the default for step is 1.

```
>>> range(3,8)
[3,4,5,6,7]
```

This is a list/sequence. As well as integers, lists can also have strings in them or a mix of integers, floats and strings. They can be created by a loop (as shown in the next section) or by explicit creation (below). Note that the `print` statement will display a string/variable/list/... to the user

```
>>> a = [5,8,"pt"]
>>> print a
[5,8,'pt']
>>> print a[0]
5
```

Loops in Python

for Loop

The general format is:

```
for variable in sequence:
     #some commands
#other commands after for loop
```

Note that the formatting (indents and new lines) governs the end of the *for* loop, whereas the start of the loop is the colon `:`.
Observe the loop below, which is similar to loops you will be using in the course. The variable `i` moves through the string list becoming each string in turn. The top section is the code in a .py file and the bottom section shows the output

```
#The following code demonstrates a list with strings
ingredientslist = ["Rice","Water","Jelly"]



for i in ingredientslist:
    print i


print "No longer in the loop"
```

```
>>>
Rice
Water
Jelly
No longer in the loop
```

while Loop

These are similar to *for* loops except they continue to loop until the specified condition is no longer true. A while loop is not told to work through any specific sequence.

```
i = 3
while i <= 15:
    # some commands
    i = i + 1 # a command that will eventually end the loop is naturally required
# other commands after while loop
```

For this specific simple *while* loop, it would have been better to do as a for loop but it demonstrates the syntax. *while* loops are useful if the number of iterations before the loop needs to end, is unknown.

The *if* statement

This is performed in much the same way as the loops above. The key identifier is the colon ： to start the statements and the end of indentation to end it.

```
if j in testlist:
    # some commands
elif j == 5:
    # some commands
else:
    # some commands
```

Here it is shown that "elif" (else if) and "else" can also be used after an *if* statement. "else" can in fact be used after both the loops in the same way.

Tuples

Tuples are basically the same as lists, but with the important difference that they cannot be modified once they have been created. They are assigned by:

```
>>> x = (4,1,8,"strng",[1,0],("j",4,"o"),14)
```

Tuples can have any type of number, strings, lists, other tuples, functions and objects, inside them. Also note that the first element in the tuple is numbered as element "zero". Accessing this data is done by:

```
>>> x[0]
4
>>> x[3]
"strng"
```

Dictionaries

A Dictionary is a list of reference keys each with associated data, whereby the order does not affect the operation of the dictionary at all. With dictionaries, the keys are not consecutive integers (unlike lists), and instead could be integers, floats or strings. This will become clear:

```
>>>x = {} # creates a new empty dictionary - note the curly brackets denoting the creation of a dictionary
>>>x[4] = "programming" # the string "programming" is added to the dictionary x, with "4" as it's reference
>>>x["games"] = 12
>>>print x["games"]
12
```

In a dictionary, the reference keys and the stored values can be any type of input. New dictionary elements are added as they are created (with a list, you cannot access or write to a place in the list that exceeds the initially defined list dimensions).

```
costs = {"CHICKEN": 1.3, "BEEF": 0.8, "MUTTON": 12}



print "Cost of Meats"
for i in costs:
    print i
    print costs[i]


costs["LAMB"] = 5


print "Updated Costs of Meats"
for i in costs:
    print i
    print costs[i]
```

```
>>>
Cost of Meats
CHICKEN
1.3
MUTTON
12
BEEF
0.8
Updated Costs of Meats
LAMB
5
CHICKEN
1.3
MUTTON
12
BEEF
0.8
```

In the above example, the dictionary is created using curly brackets and colons to represent the assignment of data to the dictionary keys. The variable i is assigned to each of the keys in turn (in the same way it would be for a list with "for i in range(1,10)"). Then the dictionary is called with this key, and it *returns the data stored under that key name*. These types of for loops using dictionaries will be highly relevant in using PuLP to model LPs in this course.

List/Tuple/Dictionary Syntax Note

Note that the creation of a: list is done with square brackets [], tuple is done with round brackets(), dictionary is done with parentheses{}.

After creation however, when accessing elements in the list/tuple/dictionary, the operation is always performed with square brackets (i.e a [3]). If a was a list or tuple, this would return the fourth element. If a was a dictionary it would return the data stored with a reference key of 3.

List Comprehensions

Python supports "List Comprehensions," which is a fast and concise way to create lists without using multiple lines. They are easily understandable when simple, and you will be using them in your code for this course.

```
a = [i for i in range(5)]
```

This statement above will create the list [0,1,2,3,4] and assign it to the variable "a".

```
odds = [i for i in range(25) if i%2==1]
```

This statement above uses the if statement and the modulus operator(%) so that only odd numbers are included in the list: [1,3,5,...,19,21,23]. (Note: The modulus operator calculates the remainder from an integer division.

```
fifths = [i for i in range(25) if i%5==0]
```

This will create a list with every fifth value in it [0,5,10,15,20]. Existing lists can also be used in the creation of new lists below:

```
a = [i for i in range(25) if (i in odds and i not in fifths)]
```

Note that this could also have been done in one step from scratch:

```
a = [i for i in range(25) if (i%2==1 and i%5==0)]
```

For a challenge you can try creating (a) a list of prime numbers up to 100, and (b) a list of all "perfect" numbers.

More List Comprehensions Examples Wikipedia: Perfect Numbers

Import Statement

At the top of all your Python coding that you intend to use PuLP to model, you will need the *import* statement. This statement makes the contents of another module (file of program code) available in the module you are currently writing i.e. functions and values defined in pulp.py that you will be required to call, become useable. In this course you will use:

```
from pulp import *
```

The asterisk represents that you are importing all names from the module of pulp. Calling a function defined in pulp.py now can be done as if they were defined in your own module.

Functions

Functions in Python are defined by: (`def` is short for define)

```
def name(inputparameter1, inputparameter2, . . .):
    #function body
```

For a real example, note that if inputs are assigned a value in the function definition, that is the default value, and will be used only if no other value is passed in. The order of the input parameters (in the definition) does not matter at all, as long as when the function is called, the parameters are entered in the corresponding correct order:

```
def listappender(list1=[0],endmessage="EOL",list2=[9,8,7]):
    list1.append(list2)
    list1.append(endmessage)
    return list1



print listappender([3,5,6],ListOver)
```

```
>>>
[3,5,6,[9,8,7],'ListOver"]
```

In the above example, the output from the function call is printed. The default value for "list1" is [0], but this was overwritten by the input of [3,5,6]. List2 of [9,8,7] was not overwritten and so the default value was used and appended, along with the overwritten "endmessage" string.

Classes

To demonstrate how classes work in Python, look at the following class structure.

The class name is Pattern, and it contains several class variables which are relevant to any instance of the Pattern class (i.e a Pattern). The first function is the `__init__` function which creates an instance of the Pattern class and assigns the attributes of name and lengthsdict using `self`..
The `__str__` function defines what to return if the class instance is printed.
The `trim` function acts like any normal function, except as with all class functions, `self` must be in the input brackets.

```
class Pattern:
    """
    Information on a specific pattern in the SpongeRoll Problem
    """
    cost = 1
    trimValue = 0.04
    totalRollLength = 20
    lenOpts = [5, 7, 9]

    def __init__(self,name,lengths = None):
        self.name = name
        self.lengthsdict = dict(zip(self.lenOpts,lengths))


    def __str__(self):
        return self.name


    def trim(self):
        return Pattern.totalRollLength - sum([int(i)*self.lengthsdict[i] for i in self.lengthsdict])
```

This class could be used as follows:

```
>>> Pattern.cost # The class variables can be accessed without making an instance of the class
1
>>> a = Pattern("PatternA",[1,0,1])
>>> a.cost # a is now an instance of the Pattern class and is associated with Pattern class variables
1
>>> print a # This calls the Pattern.__str__() function
"PatternA"
>>> a.trim() # This calls the Pattern.trim() function. Note that no input is required. The self in the function definition is an implied input
6
```

Previous: (3a) Installing Python at Home
Next: (4a) Installing PuLP at Home