Python Notebooks

For Loops

Learning Outcomes.

After studying this note book you should know:

- What a loop is.
- How to create a Python For loop
- How to loop over all the items in a list.
- How to loop over a range of numbers.
- How to use the enumerate() function in a For loop
- How to use the zip() function to loop over two lists in parallel.

What is a loop?

Loops are essential to programming, and permit a program to do the same thing over and over again some number of times. The actual number of times, or "iterations", of the loop can vary from one run to another, by being made to depend on a property of a variable.

For instance, if you want to subtract the mean from every data point in a list of data (i.e., center the data), then your program needs to traverse the list from one end to the other, subtracting the mean from successive items. For this you would use a loop.

For loops.

The most frequent kind of loop we will use is called For loop, because it begins with the word for (most programming languages have For loops, though the details of how to use them vary.) We will first see how to use a For loop with a list.

For loops on lists

SYNTAX:

> for *item* in *list*:
>
> > <Body of code. Do something to *item*>

N.B. The ":" at the end of the first line is obligatory. The terms *item* and *list* represent variables. They can have any name you want, but the latter should be a list (or some other "iterable" structure). On each iteration of the loop *item* will take on the value of the next thing in the list.

Suppose we have list called data. The For loop can traverse the list from end to end, acting on every item in the data in turn. In the following example, we simple print each item in the list:

```
#Some data in a list. Experiment with changing it.
data = [20, 46 , 57, 52, 60, 45]

for item in data:
  print(item)
```

```
20
46
57
52
60
45
```

What is happening is that the variable "item" following the word "for" is being assigned successive values from the list "data". On the first iteration of the loop item = 20, on the second iteration item = 46 and so on until item = 45. This is the end of the list, and the loop then terminates itself.

Here some more examples, using the same list:

```
#Square every item in the data
for dataPoint in data:  #we can use any variable name for the item
  print(dataPoint ** 2) #as long as we use the same name in the code
```

```
400
2116
3249
2704
3600
2025
```

```
#check for numbers greater than some threshold
thresh = 50  #threshold
for number in data:   #yet another variable name for the list items
  if number > thresh:
    print(number, 'is bigger than', thresh)
  else:
    print(number, 'is smaller than', thresh)
```

```
20 is smaller than 50
46 is smaller than 50
57 is bigger than 50
52 is bigger than 50
60 is bigger than 50
45 is smaller than 50
```

```
#Find the largest number in the data
biggest = 0
for item in data:
  if item > biggest: #if the number is larger than biggest
    biggest = item   #then make it the biggest

#NOTE the indentation level of the following line
#it is outside the loop

print('The largest number is', biggest)
```

```
The largest number is 60
```

Note how in the last example that the print command *is not inside the for loop*, becuse it is not indented with respect to it. So it is only excuted once, after the for loop has exited. Try indenting the print command to the same level as the "if" statement, and see what happens.

```
#sum the data
total = 0 #variable to hold the running total
for item in data:
  total = total + item

  print("The sum is", total) #N.B. print command is inside the loop
```

```
The sum is 20
The sum is 66
The sum is 123
The sum is 175
The sum is 235
The sum is 280
```

Note how this time the print command is inside the for loop, so it is executed on every iteration of the loop, showing the running total. Un-indent it so it is outside the loop. What should happen?

Of course, the items in the list do not have to be numbers for the loop to work. They can be anything:

```
#Greet a list of people by their names
names = ['Jack', 'Jill', 'Anne', 'Bill', 'Tom', 'Jane']

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

```
Hello Jack
Hello Jill
Hello Anne
Hello Bill
Hello Tom
Hello Jane
```

```
#bilingual gteeting
UK = [ 'Jill', 'Jane', 'Tom', 'Bill']     #British names
SP = [ 'Carlos', 'Pili', 'Juan', 'Maria'] #Spanish names
names = UK + SP #join lists

for name in names:
  if name in UK:
    greeting = 'Hello ' + name
  elif name in SP:
    greeting = 'Hola ' + name

  print(greeting)
```

```
Hello Jill
Hello Jane
Hello Tom
Hello Bill
Hola Carlos
Hola Pili
Hola Juan
Hola Maria
```

Loops a number of times with a range.

It sometimes happens that you just want a loop to iterate a fixed number of times, perhaps with an index to the iteration. In this case it is best to use the Python range command in place of the lists used above. The syntax is exacly the same:

```
#numbers and their squares from 1 to 10
#the for loop variable "index" takes on successive values from 1 to
#10 produced by the range(start, finish) method
for index in range(1,11):
  print(index,   index ** 2)
```

```
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

Getting the loop iteration index: the enumerate() method

While using a for loop over a list, you may sometimes want to know the numerical index of the current item in the list. For this, Python provides the enumerate(*list*) function, which is used with a for loop as follows:

```
poem = ['the', 'cat', 'sat', 'on','the', 'mat']
for index, word in enumerate(poem):
  print(index, word)
```

```
0 the
1 cat
2 sat
3 on
4 the
5 mat
```

On each iteration of the loop, the *two* loop variables "index" and "word" are *both* assigned values. The first of the two (it doesn't matter what it is called) is assigned *the index number of the current list item*, and the second is assigned its value (i.e., the second variable acts like the for-loop variable without enumerate()).

For instance, suppose we want to find the *location in the data* of the maximum value, e.g., finding which was the highest scoring subject in a test. We can find the maximum value (as done above) while using enumerate() to record the location:

```
#Find the location of the maximum score in the data

#Some data: Experiment with changing the location of the maximum value
data = [20, 46 , 57, 52, 60, 45, 42, 17, 32]
biggest = data[0] #variable to hold the maximum value
maxIndex = 0 #variable to hold the index to the maximum value

for index, item in enumerate(data):
  if item > biggest: #if the number is larger than biggest
    biggest = item   #then make it the biggest
    maxIndex = index #remember its associated index

#show the index to the biggest number
print('The maximum is at index', maxIndex)
#Check: use the index to get the maximum from the data
print('Its value is', data[maxIndex])
```

```
The maximum is at index 4
Its value is 60
```

Looping through two (or more!) lists in parallel: the zip() function.

Sometimes we can have two sets of data (of the same length), and we want to loop through both *in parallel*, so as to compare or manipulate the *corresponding items* of the two sets (that is, the items with same index in their respective lists). For instance, we may want to add or subtract corresponding data points. For this Python provides the zip() function.

zip(*lis1, list2*) takes two same length lists and "zips" them together so that the items at the same location in the two lists have been paired together. (Actually, you can zip as many lists together as you like, but we will stick to two!)

Here is an example of zip() acting on two lists:

```
#Some subject ID numbers and their data: Experiment by changing these.
subjects = [20, 21, 22, 24, 25, 27, 30] #subject numbers
RTs  = [1092, 1200, 1342, 987, 1234, 890, 1012] #subject reaction times

#form the zip object from the two lists
zippedData = zip(subjects, RTs)

#make a list from the zip to show the effect
print('Zipped data =', list(zippedData))
```

```
Zipped data = [(20, 1092), (21, 1200), (22, 1342), (24, 987), (25, 1234), (27, 890), (30, 1012)]
```

As can be seen, in the list formed from the zipped data, each subject number has been *paired with the associated RT*; each pair of values is shown in parentheses, e.g., (20, 1092), in which the first item of thepair (20) comes from the subjects list, and the second (1092) from the RTs list.

On a technical note: the object formed directly by zip() is not a list, it is a special "zip" object. If you try to print it, it doesn't show anything useful (try it!). So in this example, it is converted to a list before printing. In the following examples with a for loop, it is not necessary to convert the zipped data into a list, as for loop can act on the items in the "zip object" itself.

To loop through a zipped structure, there will be two loop variables (similar to enumerate, above); the first will take on the value of the first item in each pair, and the 2nd will take the value of the 2nd item in the pair. Here is a simple example using the zipped data:

```
#With zip, two loop variables are required, one for each
#item in the pairs produced by zip()
for subject, RT in zip(subjects, RTs):
  print(subject, RT)
```

```
20 1092
21 1200
22 1342
24 987
25 1234
27 890
30 1012
```

This has allowed us to display each subject number next to their RT.

The following example gets the differences between two sets of scores by subtracting corresponding elements of the two data sets:

```
#RT data from 2 conditions: Experiment with changing it.
RTs_1  = [1092, 1200, 1342, 1435, 1234, 1002, 1012]
RTs_2  = [1098, 1350, 1310, 1612, 1230, 1007, 1156]

#zip the data, so the RTs in the two sets are paired up
allRTs = zip(RTs_1, RTs_2)
print('Zipped data =',list(allRTs))

#loop through the zip
print('Differences: RT2 - RT1')
for rt1, rt2 in zip(RTs_1, RTs_2):
  print(rt2 - rt1)
```

```
Zipped data = [(1092, 1098), (1200, 1350), (1342, 1310), (1435, 1612), (1234, 1230), (1002, 1007), (1012, 1156)]
Differences: RT2 - RT1
6
150
-32
177
-4
5
144
```

Combining enumerate and zip

Just to boggle your minds, the enumerate method can act on a zip object! The for loop will need three loop variables, one for the index produced by enumerate and two for the pairs in the zip:

```
##Combining enumerate and zip
#Three loop variables are required, the first for the enumerate index,
#and the secod and third for the pairs in the zipped data
for subjectIndex, (rt1, rt2) in enumerate(zip(RTs_1, RTs_2)):
  print('Subject', subjectIndex + 1, "RTs", rt1, rt2, ' RT2 - RT1 =', rt2-rt1)
```

```
Subject 1 RTs 1092 1098  RT2 - RT1 = 6
Subject 2 RTs 1200 1350  RT2 - RT1 = 150
Subject 3 RTs 1342 1310  RT2 - RT1 = -32
Subject 4 RTs 1435 1612  RT2 - RT1 = 177
Subject 5 RTs 1234 1230  RT2 - RT1 = -4
Subject 6 RTs 1002 1007  RT2 - RT1 = 5
Subject 7 RTs 1012 1156  RT2 - RT1 = 144
```