1 Loops and iterating

for loops can be used to step through lists, tuples, and other 'iterable' objects.

Iterating through a list:

```
for item in [10,25,50,75,100]:

    print (item, item**2)

OUT:

10 100
25 625
50 2500
75 5625
100 10000
```

A *for* loop may be used to generate and loop through a sequence of numbers (note that a 'range' does not include the maximum value specified):

```
for i in range(100,150,10):

    print(i)

OUT:

100
110
120
130
140
```

A for loop may be used to loop through an index of positions in a list:

```
my_list = ['Frodo','Bilbo','Gandalf','Gimli','Sauron']

for i in range(len(my_list)):

    print ('Index:',i,', Value',my_list[i])

OUT:

Index: 0 , Value Frodo
Index: 1 , Value Bilbo
Index: 2 , Value Gandalf
Index: 3 , Value Gimli
Index: 4 , Value Sauron
```

1.1 Breaking out of loops or continuing the loop without action

Though it may not be considered best coding practice, it is possible to prematurely escape a loop with the *break* command:

```
for i in range(10): # This loop would normally go from 0 to 9

    if i == 5:

        break

    else:
```

```
        print(i)

print ('Loop complete')

OUT:

0
1
2
3
4
Loop complete
```

Or, rather than breaking out of a loop, it is possible to effectively skip an iteration of a loop with the *continue* command. This may be places anywhere in the loop and returns the focus to the start of the loop.

```
for i in range (10):

    if i%2 == 0: # This is the integer remainder after dividing i by 2

        continue

    else:

        print (i)

print ('Loop complete')

OUT:

1
3
5
7
9
Loop complete
```

1.2 Using pass to replace active code in a loop

The *pass* command is most useful as a place holder to allow a loop to be built and have contents added later.

```
for i in range (10):

    # Some code will be added here later

    pass

print ('Loop complete')

OUT:

Loop complete
```