Lists in Python

Variables

To this point, the programs written have used variables to store data. Variables store a **single** piece of data. Thedata stored in these variables can be integer or float numbers or a string.

Lists

A **list** is a data storage type which can store more than one piece of data (ie. a list or collection of data). In Python, this data can be of the same type - float , integer or string or it can be a combination of data types. Lists can also be called **arrays** or **subscripted variables**.

Like a variable, a list must have a specific name - same naming rules apply. A list is made up of a number of storage places for a single piece of data. Each of these storage spots is called an **element**. These elements are identified using an integer greater than or equal to 0 which describes its position in the list. This number is known as a **subscript** or an **index**. For example, given the following Python command:

addresses [3] = "345 Johnson"

- the name of the list is ***addresses***
- the element is *addresses **[3]*** -> pronounced '*addresses sub 3*'
- the subscript or index is ***3***
- the data stored in this element is ***"345 Johnson***"

Additional Notes on Lists

1. When creating a list in Python, it must be **initialized**" (given an initial value) using the square brackets [ ]
   Example 1: **listName = [5, 6, 7]** *# Initializes a 3 element list named listName*
   Example 2: **anEmptyList = [ ]** *# Initializes an empty list named anEmptyList*

2. Lists are used when data must be stored for later use. If the data is to be processed or outputted immediately after it is inputted, variables should be used to reduce complexity.

3. If a command calls an element with a subscript greater than the size of the list, an "out of range error" will result.

4. The first subscript of a list is 0, the second is 1, etc. This is similar to the substrings of a string.

5. Remember that lists usually contain more than one piece of data. Loops (especially for loops) are often used when working with lists.

Examples of Simple Program Using a List

*''' **Example One - Initializes a list and outputs the data in order and then in reverse order** '''*

```
letters = ["a", "b", "c", "d"] # Initializes a list called number with 4 strings

print ("The letters in order are:")
for element in letters :   # Will iterate through the list, one element at a time
        print (element)          # Outputs each element in the list

print ("The letters in reverse order are:")
for element in reversed (letters) :        # Reverses the loop
        print (element)                    # Outputs each element in the list
```

*''' **Example Two - Initializes an empty list, allow the user to input data and outputs the data after all of the names have been inputted by the user** '''*

```
aListOfNames = [ ]  # Initializes an empty list - ie. a list with no values. This must be done before using a list
while True:
        aName = input ("Input a name: ") # Stores the name inputted by the user in the variable aName
        aListOfNames.append (aName)     # The .append function adds an element to the end of the list
                                        # In this case, the name in the aName variable is added to the end of #
                                        the list
        ans = input ("Do you wish to continue? (Y or N) ")
        if ans == "N" or ans == "n":     # Note that if or is used, the entire conditions must be used
                break                    # This means you cannot type if ans == "N" or "n"
print ()
print ("The names in the list are:")        # Note this is outside the loop above
for elem in aListOfNames:        # Of course any proper variable name can be used instead of elem
        print (elem) # Outputs each element
```