www.thegordon.edu.au

Python Programming

Section 7 – Lists

UEE60411 Advanced Diploma of Computer Systems Engineering

UEENEED103A Evaluate and modify Object Oriented code programs

For further information:

Steve Gale
ICT – Science and Technology
Ph: (03) 5225 0940
Email: sgale@gordontafe.edu.au

1 – Lists

So far, when using variables we've been using single values. These are sometimes called **scalar** variables. When you're only dealing with a few variables, there is no problem with creating and using scalar variables. However, if you're dealing with hundreds, or even thousands of variables, it can quickly become unwieldy.

This is where **lists** come in. Most other languages call them **arrays**, but the principle is the same.

A **list** is basically a group of variables that all have the same name. We can refer to them by the variable name, plus an **index**, which tells the program which element of the list is being referred to.

So, for example, let's create a variable called `num`. Rather than just assigning a single value to it, we're going to assign it a list, using the following code:

```
num = [ 11, 2, 33, 14, 55 ]
```

The length of this list is 5 (there are five elements), and they are referred to by the indices 0 to 4 (the numbering starts at 0). The list contains the values:

```
11, 2, 33, 14, 55
```

If we wanted to refer to the list's values, we use the list name, plus its index in square brackets. Thus, our list's values would be:

```
num[0] → 11
num[1] → 2
num[2] → 33
num[3] → 14
num[4] → 55
```

Changing a List Element's Value

If we want to change a list element's value, we use the index (sometimes called a **subscript**) as part of the assignment statement. To change the element with the value 33 to 105, for example, we would use the code:

```
num[2] = 105
```

Our list now contains the values

```
11, 2, 105, 14, 55
```

If we wanted to copy the value of the last element into the first? We would use the following:

```
num[0] = num[4]
```

Our list now contains the values

```
55, 2, 105, 14, 55
```

Printing a List

If we want to print an item from a list, we use the `print()` function in conjunction with the list. To display the third value (105), for example, we would use:

```
print(num[2])
```

(remember, because the index starts at 0, index 2 is the *third* item in the list)

If you want to print the *whole* list, you use the `print()` function without the elements index, like so:

```
print(num)
```

This will give you the output:

```
[ 55, 2, 105, 14, 55 ]
```

You can also print from the *end* of the list by using a negative index. For example:

```
print(num[-2])
```

will display 14 on the console.

The `len()` Function

We can use the `len()` function of a list to access its length. This is extremely useful, becase we will often want to – for example – run a while loop a number of times equal to the length of the list. We might also want to change the length, which would require us to put the output of the `len()` function into a variable. Note that this returns **the number of elements within a list**, NOT the highest element number.

If we wanted to print the length of a our list as a number, we would use the following code, This would print the number “5” to the console:

```
print(len(num))
```

Removing an Element from a List

You can use the `del` command followed by the list reference to delete an element. If you wanted to remove the second element from our num list, then display the list and its length, for example, you would use:

```
del num[1]
print[len(num))
print(num)
```

The output of this would be:

```
4
[ 55, 105, 14, 55 ]
```

(The second element, `num[1]`, with the value 2, has been removed).

Adding an Element to a List

You can use the `append()` method to tack a new element onto the end of the list. For example, to add 101 onto the end of our `num` list and then display its new length and content, we would use the following code:

```
num.append(101)
print(len(num))
print(num)
```

This will give the output:

```
5
[ 55, 105, 14, 55, 101 ]
```

You can also use the `insert()` method, which is generally preferable to `append()`. It allows you to insert a new element wherever you wish, rather than just at the end of the list. It takes two arguments: *where*, and *what*. The *where* is the location within the list, and the *what* is the value to be inserted. Note that all existing elements to "the right" of the inserted one are shifted one value "to the right".

If we wanted to add 256 after 105 in our num list, and display the result, we would use the following:

```
num.insert(2, 256)
print(len(num))
print(num)
```

Output:

```
6
[ 55, 105, 256, 14, 55, 101 ]
```

What code would we use to insert the number 10 at the end of the list when we didn't know how long the list was?

Looping Through a List

Lists really come into their own when you use a looping mechanism to work your way through them, adding or deleting items as you go. Let's say you wanted to created a list called numbers that had the values 1 – 5 in it. You could do it manually with numbers[0] = 1, numbers[1] = 2, etc. This is a pretty long-winded way to do it, however, especially if you imagine a list with a hundred or ten thousand elements.

Alternatively, you could do this:

```
numbers = []
for i in range(5):
      numbers.append(i + 1)
print(numbers)
```

This will give you the result of:

```
[ 1, 2, 3, 4, 5 ]
```

The code has created a list containing those five numbers, but done so programmaticially, without you having to explicitly place a value in each one by hand.

If we wanted to add those numbers into a list in reverse order, we can use the **`insert()`** method, always ensuring each new number is placed at the beginning of the list, while forcing the other numbers "to the right" of the list, like so:

```
numbers = []
for i in range(5):
      numbers.insert(0, i + 1)
print(numbers)
```

This will give you the result of:

```
[ 5, 4, 3, 2, 1 ]
```

Using a List

We're going to create a list of integer values, then create a **`for`** loop to calculate the sum of those values. This seems trivial at first, but if you look carefully at the code you'll see that we can change the size of the list (**`numbers`**) to *anything we like*. We start with five values, but we're going to modify the program in a moment to calculate the sum of *one hundred* values without changing the actual code of the for loop at all.

```
numbers = [10, 1, 5, 3, 5]
sum = 0
for i in range(len(numbers)):
    sum += numbers[i]
print(sum)
```

This will give you the output of 24, the total of 10 + 1 + 5 + 3 + 5

Let's say we make numbers a list of all the numbers from 1 – 100. We can still use the exact same loop to give us the total output.

```
numbers = []
sum = 0
length = int(input("Enter length: "))

for x in range(1, length + 1):
    numbers.append(x)

for i in range(len(numbers)):
    sum += numbers[i]
print(sum)
```

By entering 100 at the prompt, we get a result of 5050, which is the sum of all numbers from 1 – 100, inclusive.

Always remember to test a program with known data that will create a known output. In this case, when prompted, enter 5. The output should be 15, which is the total of 1 + 2 + 3 + 4 + 5, so we know the program is working correctly.

Often when coding with lists, you will get the following error (or variations on it):

```
IndexError: list index out of range
```

This means that you have attempted to address an element of the list which is "off the end" of the list. If you have a list that has 100 elements, and you attempt to address element 101, for example, the compiler will generate an error because no such element can exist.

Always double-check your indices and your ranges to ensure you are using the correct values.

If you used the following code for the above program, for example:

```
 5  numbers = []
 6  sum = 0
 7  length = int(input("Enter length: "))
 8
 9  for x in range(1, length + 1):
10      numbers.append(x)
11
12  for i in range(len(numbers) + 1):
13      sum += numbers[i]
14  print(sum)
```

You can see in line #12 that we have told the **`for`** loop to go *past* the end of the list (its length, plus 1). Thus, in the next line, where the calculation occurs using the list, the last iteration of the **`for`** loop will try to address an element that does not exist.

`For` loops and Lists

So far we've been using the list's index to work our way through a loop. One of the nifty things in Python is that you can use the list instead of the `range(x)` statement in a `for` loop. It means that you can navigate through the list using only the index variable of the loop (`i` in the code below) and not worry about using the index of the list. Nor do you need to worry about the `len()` function.

The code below does exactly what the code on the previous page does, except you will note that the indices are now not necessary.

```
numbers = []
sum = 0
length = int(input("Enter length: "))

for x in range(1, length + 1):
    numbers.append(x)

for i in numbers:
    sum += i
print(sum)
```

Note that we still need to use a more traditional `for` loop to get the data into the numbers list. This is because there is no way of establishing the length of the list without creating the `length` variable and using it to establish the range of the loop that will initialise the list's elements.

Sorting a List

It's useful to be able to sort a list into ascending (smallest → largest) or descending (largest → smallest) order. Most modern languages have methods or functions which allow you to do this, and Python is no exception. There are two common methods used: `sorted()` and `sort()`.

sorted()

The `sorted()` method returns a list that has been sorted into either ascending or descending order. It's important to note that the value returned is **temporary** – you can iterate through it (ie: you could use it in a loop), but the `sorted()` method **does not** change the original list. Enter the following code into a Python progam to illustrate this:

```
numbers = [ 5, 2, 3, 4, 1 ]

print(sorted(numbers))

print(numbers)
```

This will generate:

```
[1, 2, 3, 4, 5]
[5, 2, 3, 4, 1]
```

The first line in the output shows the sorted version of the `numbers[]` list, but you can see in the second line that the actual list hasn't been changed.

You can add the `reverse=True` command as an extra parameter in the `sorted()` method which will sort the list into descending order. For example:

```
numbers = [ 5, 2, 3, 4, 1 ]

print(sorted(numbers, reverse=True))

print(numbers)
```

Will generate the output:

```
[5, 4, 3, 2, 1]
[5, 2, 3, 4, 1]
```

Because it's using `sorted()`, the actual `numbers` list hasn't changed, but the display in the first line now shows the list sorted into descending order.

sort()

The `sort()` method is superficially similar, but doesn't actually return a value. Instead, it **changes the list**. We can demonstrate this with the following code:

```
numbers = [ 5, 2, 3, 4, 1 ]

numbers.sort()

print(numbers)
```

This produces the following output:

```
[1, 2, 3, 4, 5]
```

Note that the actual `numbers` list is now changed – it has been resorted out of the original order. You can also used `reverse=True` as an extra parameter to sort the list into descending order.

```
numbers = [ 5, 2, 3, 4, 1 ]

numbers.sort(reverse=True)

print(numbers)
```

Outputs:

```
[5, 4, 3, 2, 1]
```

Searching a List

It's important that we are able to search through a list to find a particular item – or conversely, to determine if an item is *not* in a list. We *could* create a `for` loop that would iterate through the list, checking each individual element and returning a value depending on whether or not our searched-for item was found.

Let's say we wanted the user to enter a value to search for, and then return a message depending on whether that value was found. We could create the following program.

```
num = [ 0, 3, 12, 8, 2 ]

sought = int(input("Enter a number to search for: "))

for x in num:
    if sought == x:
        message = "Found it"
        break
    else:
        message = "Not here"

print(message)
```

If we run it, and enter 3 for the sought after number, we get:

```
Enter a number to search for: 3
Found it
```

(Note the use of the `break` command. This means that when the sought-after number is found, the `if` statement stops executing. If we don't use this, it will continue to search for 3 in every element of the list)

More simply, however, we can use the `in` or `not in` operator to test whether a value is present in the list.

In the following example, the program will return `True` or `False` depending on whether the value we are searching for is found in the list.

```
num = [ 0, 3, 12, 8, 2 ]
print(5 in num)
print(5 not in num)
print(12 in num)
```

Will produce

```
False
True
True
```

We could extend this to ask the user for a value to search for, and then produce a message depending on the result of the search.

To reproduce our original search using this technique, we could simplify it with the following:

```
num = [ 0, 3, 12, 8, 2 ]

sought = int(input("Enter a number to search for: "))

if sought in num:
    print("Found it")
else:
    print("Not here")
```

We've reduced the important part of the program from 7 lines to 4, and it is much more straightforward.

Splitting a String

Sometimes it's handy to be able to enter a long set of words that we *eventually* want to use in a list, but which would be a bit of a pain to enter one at a time. For example, let's say we wanted to enter a whole list of animals as strings – normally, we would have to enter each with enclosing quotes and separated with commas. We would have to do something like this:

```
animals = ['ant', 'baboon', 'badger', 'bat', 'bear', 'beaver', 'camel',
           'cat', 'clam', 'cobra', 'cougar', 'coyote', 'crow', 'deer',
           'dog', 'donkey', 'duck', 'eagle', 'ferret', 'fox', 'frog',
           'goat', 'goose', 'hawk', 'lion', 'lizard', 'llama', 'mole',
           'monkey', 'moose', 'mouse', 'mule', 'newt', 'otter', 'owl',
           'panda', 'parrot', 'pigeon', 'python', 'rabbit', 'ram', 'rat',
           'raven', 'rhino', 'salmon', 'seal', 'shark', 'sheep', 'skunk',
           'sloth', 'snake', 'spider', 'stork', 'swan', 'tiger', 'toad',
           'trout', 'turkey', 'turtle', 'weasel', 'whale', 'wolf', 'wombat', 'zebra']
```

You can see straight away that it would be easier if you could just enter the words, one after another (it's still pretty onerous, but it beats the quotes/commas you need to use as shown!)

By making the initialisation a single long string, with spaces separating the words, you can use a `.split()` function at the end which will split up the string into a list. Experiment with the following, entering it into a Python program (or IDLE) and then printing it out – you'll see that the string has been converted into a list.

```
animals = 'ant baboon badger bat bear beaver camel cat
 clam cobra cougar coyote crow deer dog donkey duck eagle
 ferret fox frog goat goose hawk lion lizard llama mole
 monkey moose mouse mule newt otter owl panda parrot
 pigeon python rabbit ram rat raven rhino salmon seal
 shark sheep skunk sloth snake spider stork swan tiger
 toad trout turkey turtle weasel whale wolf wombat
 zebra'.split()

print(animals)

print(animals[5])
```

Here is the output (the whole list is truncated, as it goes off the page to the right).

```
['ant', 'baboon', 'badger', 'bat', 'b
beaver
```

You can see that what was a single character string has been converted into a list, and we can use the list as normal to access elements within it.

You can also add a parameter inside the `split()` function to override what Python considers to be the **delimiter**. By default, it's a space. But if you had, say, a comma-separated list, you could set that. For example:

```
names = 'Harry, Ron, Hermione, Draco, Ginny'.split(', ')

print(names)
print()
print(names[2])
```

Running this program gives the following output:

```
['Harry', 'Ron', 'Hermione', 'Draco', 'Ginny']

Hermione
```

By putting ", " as the delimiter, Python considers the comma + space to be the separator for each item in the list. Try leaving out the space – what happens?

List Exercises

1. Create **list01.py**. It should initialise and display three lists that will print the following information.

```
['Larry', 'Moe', 'Curly', 'Bud', 'Lou']
['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December']
[1.79, 1.82, 1.65, 1.73, 1.91, 1.55, 1.62]
```

2. Create **list02.py**. It should:
   Initialise a list (called **days**) with 7 strings representing the days of the week.
   Initialise a list (called **months**) with 12 integers to represent the days in each month.
   Initialise a list (called **balance**) with 6 floats representing six bank balances.
   Display the first and last days of the week; the first, 5th, and last integer in the months list; the first, third, and last bank account balance.
   Finally, display the number of elements in each list. Your output should be:

```
First and last days: Sunday and Saturday
First, fifth, and last months: 31, 31, and 31
First, third, and last balance: 106.82, 87.67, and 256.76
days has 7 elements
months has 12 elements
balance has 6 elements
```

3. Write **list03.py**. It should use a for loop to read 10 integers from the keyboard into a list called num. It should then output the values stored in the array, one per line, as shown below:
   (HINT: Use the `append()` method)
   (NOTE: For the output loop, try using a `range()`, and then try using the name of the list – what happens?)

```
Enter integer: 1
Enter integer: 2
Enter integer: 3
Enter integer: 4
Enter integer: 5
Enter integer: 6
Enter integer: 7
Enter integer: 8
Enter integer: 9
Enter integer: 10
num[0] = 1
num[1] = 2
num[2] = 3
num[3] = 4
num[4] = 5
num[5] = 6
num[6] = 7
num[7] = 8
num[8] = 9
num[9] = 10
```

4. Save **list03.py** as **list04.py**, and modify it to read characters into the array (rename the array to **letters**). It should produce the following output.

```
Enter character: a
Enter character: b
Enter character: c
Enter character: d
Enter character: e
Enter character: f
Enter character: g
Enter character: h
Enter character: i
Enter character: j
letters[0] = a
letters[1] = b
letters[2] = c
letters[3] = d
letters[4] = e
letters[5] = f
letters[6] = g
letters[7] = h
letters[8] = i
letters[9] = j
```

5. Save **list04.py** as **list05.py**. Modify the program to display the list in reverse order.

```
Enter character: a
Enter character: b
Enter character: c
Enter character: d
Enter character: e
Enter character: f
Enter character: g
Enter character: h
Enter character: i
Enter character: j
letters[0] = j
letters[1] = i
letters[2] = h
letters[3] = g
letters[4] = f
letters[5] = e
letters[6] = d
letters[7] = c
letters[8] = b
letters[9] = a
```

6. Create **list06.py**. Initialise an array of 10 integers, called number[].
   Use a for loop to display the numbers in the list, one per line.
   Use another loop to calculate the total sum of the numbers and then display this result.
   Finally, use another loop to display the numbers in the list in reverse order.

```
1
2
3
4
5
6
7
8
9
10


The sum of the numbers is: 55

10
9
8
7
6
5
4
3
2
1
```

7. Create **list07.py**. Declare a list that will hold 5 strings representing the names of people.
   Use a for loop to populate the list via user input.
   Use another loop to display the names entered, one per line.
   Display the first, third, and last name entered.

```
Enter name: Tom
Enter name: Dick
Enter name: Harry
Enter name: Ron
Enter name: Hermione
Tom
Dick
Harry
Ron
Hermione

Tom
Harry
Hermione
```

8. Create **list08.py**. Using the `split()` function, create a list of twenty animals. The program will:
   Display the list
   Sort into alphabetical order
   Display the list
   Sort into reverse alphabetical order
   Display the list.
   Your output should be like this:

```
['zebra', 'cougar', 'dolphin', 'wombat', 'kangaroo', 'possum', 'lion', 'shark', 'porcupine', 'budgerigar', 'sloth', 'wallaby', 'cow', 'cat', 'dog', 'mouse', 'bat', 'tegu', 'elephant', 'rhinoceros']

['bat', 'budgerigar', 'cat', 'cougar', 'cow', 'dog', 'dolphin', 'elephant', 'kangaroo', 'lion', 'mouse', 'porcupine', 'possum', 'rhinoceros', 'shark', 'sloth', 'tegu', 'wallaby', 'wombat', 'zebra']

['zebra', 'wombat', 'wallaby', 'tegu', 'sloth', 'shark', 'rhinoceros', 'possum', 'porcupine', 'mouse', 'lion', 'kangaroo', 'elephant', 'dolphin', 'dog', 'cow', 'cougar', 'cat', 'budgerigar', 'bat']
```