Lab 3: Lists and Strings

CSE/IT 107

NMT Computer Science

"Each decision we make, each action we take, is born out of an intention."

— Sharon Salzberg

"Programming is learned by writing programs."

— Brian Kernighan

"The purpose of computing is insight, not numbers."

— Richard Hamming, 1962

---

1 Introduction

2 Lists

One of the most important data types in Python is the list. A list is an ordered collection of other values. For example, we might create a list of numbers representing data points on a graph or create a list of strings representing students in a class. Creating a list is simple. We simply place comma separated values inside square brackets.

```
>>> values = [1, 2, 3, 4, 5]
>>> print(values)
[1, 2, 3, 4, 5]
```

Once we put a value in a list, we can treat the list as a single entity or access individual values. In order to access an individual element, we need to use the index of the value we want to access. The index is the position of the element in the list if we start numbering the elements from 0. When we are accessing list elements by index we can use them in any way we might use a normal variable.

```
>>> values = [23, 7, 18, 0.23, 91]
>>> print(values[0])
23
>>> print(values[2])
```

```
18
>>> print(values[3])
0.23
>>> values[3] = 7.5
>>> print(values[3])
7.5
>>> print(values)
[23, 7, 18, 7.5, 91]
>>> values[0] = values[0] + values[1]
>>> print(values)
[30, 7, 18, 7.5, 91]
```

We can use an existing variable when creating a list. If we change the value of the variable, the value in the list will stay the same.

```
>>> x = 35
>>> k = 19
>>> y = 5
>>> values = [x, k, y]
>>> print(values)
[35, 19, 5]
>>> x = 1
>>> print(values)
[35, 19, 5]
```

All values in a list do not need to be the same type. If we want, we can create a list with both numbers and strings (though this is almost never done).

```
>>> values = [2, 'hello', 5.3]
>>> print(values)
[2, 'hello', 5.3]
```

If you want, you can even put a list inside a list!

```
>>> values = [1, 5, 2]
>>> more_values = [7, 'test', values]
>>> print(more_values)
[7, 'test', [1, 5, 2]]
>>> print(more_values[2])
[1, 5, 2]
```

Note that, unlike with other variables, changing an element of a list inside another list will change both values.

```
>>> values = [1, 5, 2]
>>> more_values = [7, 'test', values]
>>> print(more_values)
[7, 'test', [1, 5, 2]]
>>> values[2] = 7
>>> print(more_values)
[7, 'test', [1, 5, 7]]
>>> values = [1, 2]
>>> print(more_values)
[7, 'test', [1, 5, 7]]
```

You generally won't have to worry about this, though the reason is because we are modifying the existing value rather than create a new list. When we reassign `values`, it no longer changes the values inside inside `more_values`. This is because we are creating a new list for `values` rather than modifying the existing list.

2.1 Summary

2.2 Exercises

3 Strings

3.1 Summary

3.2 Exercises

4 For Loops Again

4.1 Summary

4.2 Exercises

5 Submitting

Files to submit:

- .py (see Section)

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should be named either `cse107_firstname_lastname_lab3.zip` or `cse107_firstname_lastname_lab3.tar.gz` depending on which method you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have `7z` installed. For Linux, look at lab 1 for instructions on how to create a tarball or use the "Archive Manager" graphical tool.

Upload your tarball or .zip file to Canvas.