Computer Science 111

Lab 6: Strings and Lists

1 Introduction

This lab shows you all the fun things you can do with strings and lists.

Start up Python and start by trying out some code in the interpreter.

References First let's get clear on how references (those arrows you draw) affect the use of these objects. Type the instructions below in the interpreter.

```
>>>alpha =  [1,2,3]
>>>beta =   [1,2,3]
>>>alpha == beta
>>>alpha is beta
```

See the difference? The boolean expression using `==` returns True because the contents are identical. The boolean expression using `is` returns False because the arrows point to two different lists.

Now try this:

```
>>>gamma = alpha
>>>gamma is alpha
>>>gamma  == alpha
```

Here, the assignment statement copies the **arrow** into gamma; it doesn't make a copy of the entire object. So now both variables are pointing to the same object.

```
>>>gamma[0] = 99
>>>gamma is alpha
>>>gamma == alpha
>>>gamma
>>>alpha
```

The first statement changes the value of `gamma`. And since both arrows point to the same object, *it also changes the value of alpha!* This is a tricky point, so be sure you understood what happened here:

- With lists, it is possible to make two variable names refer to the same list (two arrows to the same object). If you change the contents of the list, that change is reflected in both variables.
- With strings, it is also possible to make two variables refer to the same object. But since strings are immutable, you can't change their contents. So you can't observe this effect with strings.
- The simple types (int, float, bool) do not use reference arrows. So it is impossible to change one variable's value and have that change reflected in another variable's value.

Now try it with strings. As user, type **the same string value** both times.

```
>>>a = input("Enter a string:")
>>>b = input("Enter a string;")
>>>a is b
>>>a == b
```

As before, the two variables have references to different strings, but the string contents are the same.

Now try this:

```
>>>a = "he" + "llo"
>>>b = "hello"
>>>a is b
>>>a == b
```

What !!!??! This seems to contradict everything that we just said: these are two different strings but both comparisons return True. Here is the reason: when a string is constructed using constants (in quotes) and operators, Python keeps a little table of what has been constructed. It won't put duplicate entries in that table, so in this case both variables have the same reference. In the earlier example the string values came from a input statements, so they were not put in that table.

2 On to the Lab!

Download `lab6.py` from the course website, fire up IDLE, and open the program in the lab. The lab asks you to write some functions that exercise your understanding of how lists work. When you are finished submit your program in the usual way.