6 Containers I

Key terms: slice `argv in len list range str tuple`

Reading: Severance 6, 8, and 10

Exercise: Write a program that prints out the last character of the first command line argument.

Frequently our models include not two or three values, but fifty, or fifty thousand. In many cases, the data in such containers can be looked up with integer indices, e.g. `names[6]`. The built-in function `len` tells us how many elements are in a container. Presently we consider the two similar classes `list` and `tuple`. They can store objects of any type at all, including more collections (thereby creating a multidimensional structure).

The `list` is the more flexible and familiar class, a mutable type which resizes as needed. It supports sequential search (`find`) and stack behavior (`append` and `pop`). A `tuple` is much more limited, because it forbids changing any of its own contents after creation. This restriction has two advantages: significantly better performance than a list, and a valuable guarantee that the object will never change. A list or tuple are easy to create from each other as needed with `list()` or `tuple()`.

```
# Demonstrate handling a list of prices.
# Note the floating-point inaccuracy in the sum.

prices = [ 1.99, 0.49, 2.99 ]
print(sorted(prices))
print(sum(prices))
```

The `str` class, short for "string", contains text. Command line arguments are available in a `list` of strings called `sys.argv`. These are the parameters that you add after a program's name in a shell command. As long as `sys` has been imported they are available. The analogous use of `input()` to prompt the user cannot substitute for `sys.argv`. `input()` is useful in interactive programs, but we want to write batch programs. Here is the `capitalize.py` program executed in the "Interpreter" week:

```
# Capitalize the first command line argument; expect an IndexError if one is not
    passed

import sys
print(sys.argv[1].upper())
```

There are several built-in functions for examining a collection at one go: `len` shows its magnitude; `max` finds its greatest value; `sum` adds its values, if possible; `any` is `True` if any value is `True`, as opposed to `all`. The operator `in` searches through a collection for a certain value: `'extro' in 'ambidextrous'` is `True`.

In addition to accessing a single element with the traditional index syntax `group[n]`, Python allows subsets of the sequence to be called for using "slicing" and "striding" notation. `group[n:m:o]`, retrieves every `o`th value from `group` between indices `n` (included) and `m` (excluded). All three numbers are optional. NB the special meaning of negative indices; they are relative to the end of the sequence.

```
# Show an ordered list of New York / San Francisco Giants championships

wins = [1905, 1921, 1922, 1933, 1954, 2010, 2012, 2014]
print ("The Giants first won the World Series in {}.".format(wins[0]))
print ("Recent wins were in {}.".format(wins[-3:]))
```

Next, we will address unordered containers.