

CSSE 120 – Introduction to Software Development

Concept: *Accumulating Sequences*

This lesson puts together three concepts that you have seen:

- *Sequences*, for example *lists* and *strings*
- The + operator as *concatenation*
- The *Accumulator Pattern (counting, summing, in graphics)*

to allow you to *accumulate (that is, “build up”) sequences*.

Example 1
(lists):

```
seq = []
for k in range(10):
    seq = seq + [k ** 2]

print(seq)
```

prints

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

The pattern for *building up (accumulating) lists* is (as shown in the above example):

1. *Before* the loop, initialize the list variable (the “accumulator”) to the empty list `[]`.
2. *Inside* the loop, include a statement of the form:

```
seq = seq + [new item to append to the sequence]
```

where `seq` is the list variable that you have chosen. The + operator is concatenation (not addition) since the arguments are sequences (not numbers). Just as for the summing pattern, it is critical to have the SAME variable on each side of the assignment.

3. *After* the loop, use the (built-up) list as desired.

Note the similarity to the *summing* pattern.

Example 2 (strings):

```
s = ''
for k in range(10):
    s = s + str(k ** 2)

print(s)
```

prints

```
0149162536496481
```

Recall that the built-in function `str` returns a string version of its argument, so the + operator is again concatenation (here, of strings).

As the example shows, the pattern for strings is identical to the pattern for lists; the difference is only that you must append strings to strings (or lists to lists).

Of note: The above technique for building up sequences is grossly inefficient in its use of time and space, in that it repeatedly builds new sequences instead of re-using space allocated once at the beginning. When we study *mutation* (soon), we’ll see a better technique.