5.9: MAXIMUM AND MINIMUM LOOPS

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

To find the largest value in a list or sequence, we construct the following loop:

CODE 5.9.1 (PYTHON):

```
%%python3

largest = None
print('Before:', largest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if largest is None or itervar > largest :
        largest = itervar
    print('Loop:', itervar, largest)
print('Largest:', largest)
```

run restart

When the program executes, the output is as follows:

```
Before: None
Loop: 3 3
Loop: 41 41
Loop: 12 41
Loop: 9 41
Loop: 74 74
Loop: 15 74
Largest: 74
```

The variable `largest` is best thought of as the "largest value we have seen so far". Before the loop, we set `largest` to the constant `None`. `None` is a special constant value which we can store in a variable to mark the variable as "empty".

Before the loop starts, the largest value we have seen so far is `None` since we have not yet seen any values. While the loop is executing, if `largest` is `None` then we take the first value we see as the largest so far. You can see in the first iteration when the value of `itervar` is 3, since `largest` is `None`, we immediately set `largest` to be 3.

After the first iteration, `largest` is no longer `None`, so the second part of the compound logical expression that checks `itervar > largest` triggers only when we see a value that is larger than the "largest so far". When we see a new "even larger" value we take that new value for `largest`. You can see in the program output that `largest` progresses from 3 to 41 to 74.

At the end of the loop, we have scanned all of the values and the variable `largest` now does contain the largest value in the list.

To compute the smallest number, the code is very similar with one small change:

CODE 5.9.1 (PYTHON):

```
%%python3

smallest = None
print('Before:', smallest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if smallest is None or itervar < smallest:
```

```
        smallest = itervar
    print('Loop:', itervar, smallest)
print('Smallest:', smallest)
```

run restart

Again, `smallest` is the "smallest so far" before, during, and after the loop executes. When the loop has completed, `smallest` contains the minimum value in the list.

Again as in counting and summing, the built-in functions `max()` and `min()` make writing these exact loops unnecessary.

The following is a simple version of the Python built-in `min()` function:

```
def min(values):
    smallest = None
    for value in values:
        if smallest is None or value < smallest:
            smallest = value
    return smallest
```

In the function version of the smallest code, we removed all of the `print` statements so as to be equivalent to the `min` function which is already built in to Python.