CSSE 120 – Introduction to Software Development

Concept: *Overloading the + Symbol*

In English and other natural languages, one word often has several different meanings. For example, consider the word "*bark*":

- The dog's *bark* woke me up.
- The aspen tree's *bark* was a silver gray.

One word – *bark* – but two completely different meanings! We determine the meaning (i.e., the ***semantics***) of the word *bark* from the context in which it is used.

In ***programming languages***, we say that a symbol is ***overloaded*** if it has two or more meanings that are distinguished by the context in which the symbol is used. The **plus symbol + is overloaded** as follows:

- When its operands are *numbers*, **+** means ***addition***.
  For example:

  `5 + 3` *evaluates to the number* `8`

  `7 + 5 + 1` *evaluates to the number* `13`

- When its operands are *sequences*, **+** means ***concatenation*** (i.e., "stitching together" two things, one after the other).
  For example:

  `[4, 3] + [1, 7, 2, 4]`
  evaluates to the list `[4, 3, 1, 7, 2, 4]`

  `(4, 1, 7) + (3, 3)`
  evaluates to the tuple `(4, 1, 7, 3, 3)`

  `'hello' + 'Dave' + '55' + '83'`
  evaluates to the string `'helloDave5583'`

That is, for sequences, the *plus* operator constructs a ***new*** sequence that has the elements of the first sequence ***followed by*** the elements of the second sequence. If the sequences are lists, the result is a list; if tuples, then a tuple; if strings, then a string, etc.

Overloaded means one symbol is "loaded" with more than one meaning. For example, the **+** operator means either:

44 + 9 → 53 (Addition)

'44' + '9' → '449' (Concatenation)

Here is one application of string concatenation:

Previously, you have seen that you can print several items on a single line by putting them in a single ***print*** statement, and you may have noticed that the ***print*** statement puts a space between each item when it prints them. The following example shows ***another*** way to print several items; this new way allows you more control.

```
x = 51
y = 3
z = 40
print(x, y, z)
print(str(x) + str(y) + str(z))
print(x + y + z)
```

The built-in ***str*** function returns a string version of its argument.

Here is the output that the code to the left produces.

```
51 3 40
51340
94
```

The built-in **str** function returns a string version of its argument – for numbers, that means the digits (as characters) stitched into a string (i.e., sequence of characters). It is similar to (but the inverse of) the ***int*** and ***float*** functions that return integer and floating-point versions of their string arguments.

Make sure that you understand ***why***:

1. The first and second of the above ***print*** statements print the same thing, except that the output from the first ***print*** statement includes spaces while the output from the second one does not.
2. The second and third ***print*** statements compute (and hence print) completely different things.