CSSE 120 – Introduction to Software Development (Robotics section)

Concept: *Sequences and Indexing*

What is a sequence?

A ***sequence*** is a type of thing in Python that represents a finite, ordered ***collection*** of things ***indexed*** by whole numbers. For example:

- A ***list***: `['red', 'white', 'blue']`
- A ***string***: `'Check out Joan Osborne, super musician'`
- A ***tuple***: `(800, 400, 310)`

There are also types for *unordered* collections, for example, ***sets*** and ***dictionaries***.

Lists, ***strings*** and ***tuples*** are all ***sequences***. Lists and tuples can contain anything; strings contain only characters. Tuples act just like lists except tuples are not ***mutable*** – we'll talk more about mutability later.

Why are sequences important?

Sequences are powerful because they let you refer to an entire collection, as well as the items in the collection, using a *single name*.

- You can still get to the ***items*** (aka ***elements***) in the collection, by ***indexing***:

```
colors = ['red', 'white', 'blue']
      colors[0]   has value  'red'
      colors[1]   has value  'white'
      colors[2]   has value  'blue'
```

Indexing starts at ZERO, not at one

The number (or variable) inside the square brackets is called the INDEX.

- And you can ***loop ("iterate") through the items*** in the collection, as in this example that constructs circles with colors taken from a list.

```
colors = ['red', 'white', 'blue', ...]
for k in range(len(colors)):
    circle = zg.Circle(...)
    circle.setFill(colors[k])
```

The* `len` *function returns the LENGTH of the sequence, that is, the number of items in the sequence.

Be sure that you understand the use of the index **k** in the above example. It is not a "magic" symbol; it is just an ordinary variable that goes **0, 1, 2, ...** per the **range** statement. Do you see now why the **range** statement is defined to start at **0** and ends one *short* of the value of its argument?

When you don't need the index itself, here is an alternative notation for looping (iterating) through a sequence:

```
for color in colors:
    circle = zg.Circle(...)
    circle.setFill(color)
```

Accessing the *last* element of a sequence – avoid this gotcha

A common ***error*** when trying to refer to the last element of a sequence is to be off-by-one, as in this example:

```
cool_words = ['aplomb', 'eviscerate', 'pataflafla', 'tmesis']

last_word = cool_words[3]                       # Correct!
last_word = cool_words[len(cool_words) - 1]     # Correct!

last_word = cool_words[4]                       # WRONG!!!
last_word = cool_words[len(cool_words)]         # WRONG!!!
```

Note!

The wrong statements above generate an error message:

```
IndexError: list index out of range
```

Cool words taken from:
www.vocabula.com/vrbestwords.asp

Different types of sequences: *list*, *string*, *tuple* and *range*

The sequence types that we will use most often are ***list***, ***string*** (***str***) and ***tuple*** (and also ***range***, but we will use ***range*** only in ***for*** statements).

- A ***list*** can contain objects of any type and is *mutable*. Literals are written using square brackets:

```
[45, 87.0, zg.Point(400, 30)]
```

- A ***tuple*** can also contain objects of any type, but is *not* mutable. Literals are written using parentheses:

```
(45, 87.0, zg.Point(400, 30))
(57,)
```

Notation for a SINGLE item in a tuple. Also, for tuples with more than one item you can omit the parentheses.

- A ***string*** can contain only Unicode characters and is *not* mutable. Literal strings are written using single or double quotation marks (either is fine):

```
'hi there'     "what's up doc?"
```

We'll discuss ***mutability*** in a forthcoming session.

An example:

Here (on the right) is an example that combines the ***summing*** pattern with the ***iterating-thru-a-sequence*** pattern.

```
def sum_all(sequence):
    """ Returns the sum of all the numbers in the
        given sequence.  Precondition: The argument
        is a sequence containing only numbers.
    """
    total = 0
    for k in range(len(sequence)):
        total = total + sequence[k]

    return total
```