STRINGS AND STRING PROCESSING

COMP130 – INTRODUCTION TO COMPUTING

DICKINSON COLLEGE

STRINGS

- A `String` is an object representing an ***ordered sequence of characters***.
  - `s = 'Hello COMP 130!'`
- Individual characters can be accessed using an *integer value* ***offset*** (a.k.a. an ***index***).
  - `s[0]` is 'H'
    - Note: Offsets/indices are ***zero based*** (i.e. they start with 0).
  - `s[4]` is 'o'
  - `funny = s[2] + s[4] + s[3]`

Offset (or index)

- Accessing an index that does not exist causes an ***Index Error***:
  - `oopsie = s[15]`

```
IndexError: string index out of range
```

TRAVERSAL

- A common pattern of computation with a sequence (e.g. a `String`) is to ***traverse*** it. That is, to use an offset (index) to access the individual items in the sequence one-by-one.

```
s = "Traverse this."
index = 0
while index < len(s):
    letter = s[index]
    print(str(index) + " : " + letter)
    index = index + 1
```

```
0 : T
1 : r
2 : a
3 : v
4 : e
5 : r
6 : s
7 : e
8 :
9 : t
10 : h
11 : i
12 : s
13 : .
```

- By changing the *setup*, *condition* and *update* the `String` can be traversed in other orders (reverse, every other character, etc.)

TRAVERSAL WITH THE `FOR IN` LOOP

- The most common case is to traverse the characters in a sequence (e.g. a `String`) from the the beginning to the end. In Python the `for in` loop simplifies this processing.
  - The *loop variable* is assigned each character in turn and can be used in computations in the *loop body*.

```
s = "As Soon As Possible."

acronym = ''
for letter in s:
    if letter <= 'Z' and letter >= 'A':
        acronym = acronym + letter

print(acronym)
```

Loop Variable

STRING SLICES

- A ***slice*** is a contiguous (adjacent) sub-sequence of the elements of a sequence.

```
motto='A useful education for the common good.'
#      0         1         2         3
#                0         0         0

sliceIt=motto[2:8]                          # useful
diceIt=motto[34:38]                         # good
diceIt=motto[len(motto)-5:len(motto)-1]     # good
start=motto[:18]                            # A useful education
end=motto[23:]                              # the common good.
```

  - The notation [start:end] specifies slice of a `String` that begins at offset `start` and ends at offset `end-1`.

STRINGS CANNOT BE CHANGED

- `Strings` are ***immutable objects***. Once they are created, they cannot be changed.
  - `s = 'Hello COMP 130!'`
    `s[0] = 'M'`
    `s[5:9] = 'CS'`
    → `TypeError: 'str' object does not support item assignment`
- Instead create a new `String` by concatenating parts of the old string with the new parts:
  - `ms = 'M' + s[1:]` `# Mello COMP 130!`
    `cs = s[:6] + 'CS' + s[10:]` `# Hello CS 130!`

SEARCH PATTERN

- To ***search*** a sequence (e.g. a `String`) is to *traverse* it looking each of the items for an item or items that meet specific criteria.
  - E.g. Find the first 'A', or the first digit, or the last space, or …

```
def find_digit(line):
    """ Return the index of the first numeric digit in the
        the given string, or -1 if line contains no digits.
    """
    index = 0
    while index < len(line):
        char = line[index]
        if char >= '0' and char <= '9':
            return index
        index = index + 1
    return -1
```

Go through the items in the sequence

Check if this item is what we are searching for. If so return where we found it.

This Item was not what we are searching for (otherwise we would have returned) so check the next one.

We have checked all of the items and didn't find a match.

AGGREGATE PATTERN

- To compute an ***aggregate***, traverse the entire sequence, compute the desired information as you go (max, min, average, count, …) and return the result.

```
def count_vowels(line):
    """ Count the number of vowels that appear in the line.
    """
    count = 0
    for char in line:
        if char=='A' or char=='E' or char=='I' or char=='O' or char=='U':
            count = count + 1
    return count
```

Setup to compute the aggregate value.

Go through each item in the sequence (one-by-one) in this case.

Compute the aggregate.

All items have been processed so return the result.