CptS 111, Spring 2023
Lect. #21, Apr. 10, 2023
Class Notes

---

Today's Agenda:

1. Some coding questions from the exam
2. Two more string methods
3. Use of `in` with string
4. String slicing

---

**Ch. 8 (cont.)**

**Strings (cont.)**

**1. Some Coding Questions from the Exam**

In [1]:
```
# Problem 31 required you to open and close files
# for reading and writing.  The for-loop was
# similar to Lab 9, Task 2:

file_in = open('poem.txt')
file_out = open('new_poem.txt', 'w')
for line in file_in:
    print(line, file=file_out, end='') # or end='', file='file_out'
    print('Ta-da!', file=file_out)
file_in.close()
file_out.close()
```

In [2]:
```
# Problem 33 was almost identical to zCA 6.11.3
# which I did in a lecture.  It just added made
# two rows.

for i in range(2):
    for j in range(3):
        print(f'{i+1}{chr(65+j)}', end=' ')
    print()
```

```
1A 1B 1C
2A 2B 2C
```

In [3]:
```
# Problem 34 was similar to PA 5 and tasks in Lab 9

def get_goodies():
    goodies = []
    while True:
        goodie = input('Enter goodie [return to stop]: ')
        if goodie == '':
            break
        goodies.append([goodie, 0])
    return goodies
```

Problem 35 was just `get_nums()` from Lab 9, Task 3, but just the three lines of the `for` -loop, not the function.

**2. Two More String Methods**

Next, we'll consider two methods that are surprisingly useful.

***D. The `.split()` Method***

- **`.split()` : creates a *list* from a *string***, splitting the string on whitespace by default, but any character or characters can be used

In [4]:
```
# .split() creates a list from a string
# If nothing is used inside the parentheses, .split() will split on
# whitespace, i.e., spaces, tabs, and newlines.

s = 'Spontaneity has its\t\t time and  \nplace.'
s.split()
```

Out[4]:
```
['Spontaneity', 'has', 'its', 'time', 'and', 'place.']
```

We see that the string has been split on the whitespace between words. We can split on entire words, on punctuation, or on individual letters.

***Importantly, whatever is used for the split is removed from the string!*** For example,

In [5]:
```
# When we split on a character (including whitespace), it's deleted

s.split('t')  # Put what you want to split on inside parens!
```

Out[5]:
```
['Spon', 'anei', 'y has i', 's\t\t ', 'ime and  \nplace.']
```

As we can see, all the `t's` have disappeared, but all the whitespace is retained. Recall that I gave you the following line to use in zyLab_PA2, our Running Time Calculator program.

```
mm, ss = int(pace.split(':')[0]), int(pace.split(':')[1])
```

You've now studied everything you need to understand this statement! Let's dissect it to understand it thoroughly.

In [6]:
```
# We start by asking a user for the pace:

pace = input('Enter pace [mm:ss]: ')  # pace is a string
```

```
Enter pace [mm:ss]: 7:30
```

In [7]:
```
# Next, let's split `pace` on the colon:

plist = pace.split(':')
print(plist)
```

```
['7', '30']
```

In [8]:
```
# We now have a list of two string elements.
# Convert the first element to an integer:

int(plist[0])
```

Out[8]:
```
7
```

In [9]:
```
# And let's make sure we really do have an integer!

type(int(plist[0]))
```

Out[9]:
```
int
```

In [10]:
```
# Putting all this together and using simultaneous assignment, we have:

mm, ss = int(pace.split(':')[0]), int(pace.split(':')[1])
print(f'I never ever ran a mile in {mm} minutes and {ss} seconds.  Oh wel
```

```
I never ever ran a mile in 7 minutes and 30 seconds.  Oh well.
```

Note that if we split on a character that begins or ends a string, we'll end up with an empty string. We always get two items on either side of whatever we split on.

In [11]:
```
# Splitting on a character that begins a string

chained = '.lower().title()'
unchained_list = chained.split('.')
print(unchained_list)
```

```
['', 'lower()', 'title()']
```

`.split()` can be useful for removing newlines from text (and recall that we can't use the string method `.strip()`). Consider the following example. Suppose we want all the numbers to be on a single line.

In [12]:
```
# Consider a string with newline characters in it

s = '0123\n4567\n8910'
print(s)
```

```
0123
4567
8910
```

In [13]:
```
# We can split on the newline character '\n'

print(''.join(s.split('\n')))
```

```
012345678910
```

In [14]:
```
# We can also use the default because newlines are whitespace

print(''.join(s.split()))
```

```
012345678910
```

In the example above, we split the string `s` on the newlines (`'\n'`). This created a list of three 4-character strings ('0123', '4567', '8910'). We then joined these strings together using the `.join()` method, which is a segue to our next method.

***E. The `.join()` Method***

- `.join()` is another useful method which can be used very effectively with `.split()`
- `.join()` takes a list as its attribute (argument) and joins the elements in the list by whatever string is desired.

The `.split()` and `.join()` methods are like inverses of each other:

- `.split()` ***creates a list from a string***
- `.join()` ***creates a string from a list***

In [15]:
```
# Example 1: Add newlines between elements of a list with .join(),
# creating a string

num_list = ['0123', '4567', '8910']
num_string = '\n'.join(num_list)
num_string
```

Out[15]:
```
'0123\n4567\n8910'
```

As we see, we can join the elements of a list by a newline. Next, consider the following string of names.

In [16]:
```
# Create a string of names

names = 'Monroe, Marilyn; Ridley, Daisy; Berry, Halle; Chastain, Jessica'
names
```

Out[16]:
```
'Monroe, Marilyn; Ridley, Daisy; Berry, Halle; Chastain, Jessica'
```

Suppose we want to separate the names into a list.

In [17]:
```
# Example 2: Separate string of names into a list by splitting on ';'

name_list = names.split(';')
name_list
```

Out[17]:
```
['Monroe, Marilyn', ' Ridley, Daisy', ' Berry, Halle', ' Chastain, Jess
ica']
```

However, note that when we split on the semicolon, we only removed it and not the space before the three last elements in the list. Let's remove the space, too.

In [18]:
```
# Revised Example 2: Remove the space after the semicolon by splitting on
# both

name_list = names.split('; ')
name_list
```

Out[18]:
```
['Monroe, Marilyn', 'Ridley, Daisy', 'Berry, Halle', 'Chastain, Jessic
a']
```

Ta da!

Finally, recall that we can use a `for` -loop to string together words, but let's see how more easily this can be accomplished using `join()`.

In [22]:
```
# Example 3: Use .join() to insert character between list elements

words = ['fly', 'by', 'night']
s = '-'.join(words)
s
```

Out[22]:
```
'fly-by-night'
```

`.split()` and `.join()` can also be used together to determine the number of visible characters there are, for example, in a sentence.

In [23]:
```
# Use .split() and .join() to determine number of visible characters
# in string; split on whitespace and use .join() with an empty string

s = 'This is a one line proof if we start sufficiently far to the left.'
len(''.join(s.split()))
```

Out[23]: 53

### 3. Use of `in` in Strings

You'll recall that you've used the Python keyword `in` when coding `for` -loops, but as mentioned in your textbook, `in` together with its counterpart `not in` can be used to determine whether a string or substring is part of another string. This is analogous to our use of `in` in PA #5 where we used `in` to determine whether a movie or video game title is in our dictionary.

In [24]:
```
# Use in to determine whether a sequence of characters is in a string

dna = 'actggtcaaaccttggtatacgtc'
if 'tata' in dna:
    print('tata is in dna.')
```

```
tata is in dna.
```

### 4. String Slicing

Sometimes we want to be able to "cut out" a piece of a string

Template for string slicing:

```
<string>[start : stop : increment]
```

where:

```
start:      slice begins at start index
stop:       slice ends one before stop index
increment:  default is 1, but other values can be chosen
```

Notes:

1. `[ : stop]` will start at 0
2. `[start : ]` will start at `start` and end at the end of the string

String slicing can be very useful. For example, in bioinformatics we can create DNA sequences from a genome. Let's say, for example, that a gene is identified by the sequence `TATA` 30 nucleotides (there are four different nucleotides: A, T, G, C) before the starting point of a gene. Then to locate the actual gene, we can do the following:

1) First split the genome on spaces and join the nucleotides together into a continuous string:

In [25]:
```
# Split genome on spaces and join together using '' to create continuous
# string

genome = 'gaaaaggcaaccgtaggggg tgatagtccc gtacaggtaa gaaggttgca gatccttga
tagggcggggcacgtggaac cctgtctgaa tttaggggga ccatcctcta aacctaagta\
ctccttagcgaccgatagtg aacaagtacc gtgagggaaa ggtgaaaaga accccggtga\
ggggagtgaaatagaacctg aaatcgggtg cttacaatca gttggagctt gtgtaggttt\
tgcagcttgctgtaattcct gcatgggtga cagcgtacct tttgcataat gggtcagcga\
gttaatctgttgagcaagct taagccgtta ggtgtaggcg tagcgaaagc gagtctgaat\
agggcgttttagttcaatgg attagacccg aaaccaagtg atctagccat ggccaggttg\
aaggtgtagtaaagtacatt ggaggaccga acctgttact gttgcaatag tattggatga\
gctgtggctaggggtgaaag gccaatcaaa cttggaaata gctggttctc cgcgaaatct\
atttaggtagagcgttgtac tataagttgc cgggggtaga gcactgaatg gactagggag\
actcatcctcttaccaaatc caatcaaact ccgaataccg gtcaactgtt ctacagcaga\
cacactgcgggtgctaagtc cgtggtggaa agggaaagag cccagatcgc cgtctaaggt\
cccaaagttatggctaagtg tggaaggagg tgagaagacc aatacagcta gtaggttggc\
ttagaagcagccatccttta aagaaagcgt aacagctcac ttgtctagat ttaagtcttc\
ttgcgccgaagatgtaacgg ggctaaagcc atacgccgaa gacgcgagtg cccggcttgc\
cgggtgcggtagcggagcgt ttcgtaagcc tgtgaaggtg gcccgtgagg gttgctggag\
gtatcgaaagtgagaatgct gacatgagta gcgtaaagga gtgtgaaaac cactcccgcc\
gaaaacctaa'
genome = ''.join(genome.split())
genome
```

Out[25]:
```
'gaaaaggcaaccgtaggggtgatagtcccgtacaggtaagaaggttgcagatccttgagtagggcgggg
cacgtggaaccctgtctgaatttaggggaccatcctctaaacctaagtactccttagcgaccgatagtga
acaagtaccgtgagggaaaggtgaaaagaaccccggtgaggggagtgaaatagaacctgaaatcgggtgct
tacaatcagttggagcttgtgtaggttttgcagcttgctgtaattcctgcatgggtgacagcgtacctttt
gcataatgggtcagcgagttaatctgttgagcaagcttaagccgttaggtgtaggcgtagcgaaagcgagt
ctgaatagggcgttttagttcaatggattagacccgaaaccaagtgatctagccatggccaggttgaaggt
gtagtaaagtacattggaggaccgaacctgttactgttgcaatagtattggatgagctgtggctaggggtg
aaaggccaatcaaacttggaaatagctggttctccgcgaaatctatttaggtagagcgttgtactataagt
tgccgggggtagagcactgaatggactagggagactcatcctcttaccaaatccaatcaaactccgaatac
cggtcaactgttctacagcagacacactgcgggtgctaagtccgtggtggaaagggaaagagcccagatcg
ccgtctaaggtcccaaagttatggctaagtgtggaaggaggtgagaagaccaatacagctagtaggttggc
ttagaagcagccatcctttaaagaaagcgtaacagctcacttgtctagatttaagtcttcttgcgccgaag
atgtaacggggctaaagccatacgccgaagacgcgagtgcccggcttgccgggtgcggtagcggagcgttt
cgtaagcctgtgaaggtggcccgtgagggttgctggaggtatcgaaagtgagaatgctgacatgagtagcg
taaaggagtgtgaaaaccactcccgccgaaaacctaa'
```

2) Next find the index for the location of `tata` :

In [26]:
```
# Use .find() method to find index for first occurrence of 'tata'

genome.find('tata')
```

Out[26]:
```
560
```

3) Finally, isolate the gene based on the location of the `tata` string. Assume the gene occurs 30 nucleotides after the tata box. Then to isolate the gene, we add 30 from the location of `tata` box and use this as the start index for the slice.

In [27]:
```
# Use string slicing to isolate gene which occurs 30 nucleotides after
# start of tata

gene = genome[560 + 30 : ]    # We can use an arithmetic operation as a
gene                          # start, stop, or increment value
```

Out[27]:
```
'gactagggagactcatcctcttaccaaatccaatcaaactccgaataccggtcaactgttctacagcaga
cacactgcgggtgctaagtccgtggtggaaagggaaagagcccagatcgccgtctaaggtcccaaagttat
ggctaagtgtggaaggaggtgagaagaccaatacagctagtaggttggcttagaagcagccatcctttaaa
gaaagcgtaacagctcacttgtctagatttaagtcttcttgcgccgaagatgtaacggggctaaagccata
cgccgaagacgcgagtgcccggcttgccgggtgcggtagcggagcgtttcgtaagcctgtgaaggtggccc
gtgagggttgctggaggtatcgaaagtgagaatgctgacatgagtagcgtaaaggagtgtgaaaaccactc
ccgccgaaaacctaa'
```

This gene example is simplified, but it gives you an idea of something real that can be done using the `.split()`, `.join()`, and `.find()` methods together with string slicing!