_C7-1_90_ThinkPython-Chap8-Theory

April 10, 2020

1 8. Strings

2 8.1 A string is a sequence

In [1]:
```
# A string is a sequence of characters
fruit = 'banana'
letter = fruit [1]
print letter #Is it the first letter of the word?
```

```
a
```

In [2]:
```
# You can use any expression, including variables and operators,
# as an index, BUT the value of the index has to be an INTEGER.
letter [1.5]
print letter
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-2-c4fb84776dbb> in <module>()
  1 # You can use any expression, including variables and operators,
  2 # as an index, BUT the value of the index has to be an INTEGER.
----> 3 letter [1.5]
  4 print letter


TypeError: string indices must be integers, not float
```

3 8.2 len

In [4]:
```
# The function len is a buit-in function that returns
# the number of characters in a string.
fruits = 'mere si pere'
print len(fruits)
```

```
12
```

```
In [5]: last = fruit [len(fruit)]
        print last


        ---------------------------------------------------------------------------

        IndexError                                Traceback (most recent call last)

        <ipython-input-5-72b03393d89e> in <module>()
    ----> 1 last = fruit [len(fruit)]
          2 print last


        IndexError: string index out of range
```

```
In [7]: #Negative indices count BACKWARD from the end of the string.
        fruct = 'bananA'
        last = fruct[-1]
        print last
```

```
A
```

4 8.3 Traversal with a for loop

```
In [13]: """
         TRAVERSAL pattern
         Having a sequence of elements, a common processing consists of:
             - starting at the beginning
             - select each element in turn
             - do something to it
             - continue UNTIL the end
         """
         index = 0
         while index < len(fruit): # vs. index <= len(fruit)
             letter = fruit[index]
             print letter, ' ',
             index = index + 1
```

```
b   a   n   a   n   a
```

4.0.1 Exercise 8.1. (page 72)

Write a function that takes a string as an argument and displays the letters backward, one per line.

```
In [10]: def displayBack1(secv):
             index = 0
             l = len(secv)
             while index < l:
                 print secv[l-1-index]
                 index = index + 1

         displayBack1('banana')
```

```
a
n
a
n
a
b
```

The same problem but using negative indices...

```
In [11]: def displayBack2(secv):
             index = 1
             while index <= len(secv):
                 print secv[-index]
                 index = index + 1

         displayBack2('banana')
```

```
a
n
a
n
a
b
```

```
In [12]: #Traversal with a for loop
         for char in fruit:
             print char, ' ',
```

```
b   a   n   a   n   a
```

```
In [14]: prefixes = 'bcdflmnptvz'
         suffix = 'or'
         for letter in prefixes:
             print letter + suffix
```

```
bor
cor
```

```
dor
for
lor
mor
nor
por
tor
vor
zor
```

4.0.2 Exercise 8.2. (page 73)

– not interesting – (It is about the English words with prefixes).....ADAPT TO Romanian?

5 8.4 String slices

```
In [13]: # A sequence of a string is called a slice.
         s = 'Monty Python'
         print s[0:5]
```

```
Monty
```

```
In [14]: print s[6:12]
         #Where is the "blank" (space) from the given string?
```

```
Python
```

```
In [3]: """
        The slicing operator [n:m] returns the part of the string
        from the n-th to the m-th INCLUDING the first,
        but EXCLUDING the last (!!!)
        """
        fruit = 'banana'
        print fruit[:3]
```

```
ban
```

```
In [4]: print fruit[3:]
```

```
ana
```

```
In [19]: print fruit [3:3]
         # This prints the void string '' !!!
```

5.0.3 Exercise 8.3. (page 74)

Given that "fruit" is a string, what does fruit[:] mean?

```
In [20]: print fruit [:]
```

```
banana
```

6 8.5 Strings are immutable

```
In [21]: greeting = 'Hello, world!'
         greeting [0] = 'J'
         print greeting
```

```
        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)

        <ipython-input-21-c7b43b81f057> in <module>()
          1 greeting = 'Hello, world!'
    ----> 2 greeting [0] = 'J'
          3 print greeting


        TypeError: 'str' object does not support item assignment
```

```
In [23]: # It follows that string are immutable -
         # you can't change an existing string
         greeting = 'Hello, world!'
         new_greeting = 'J' + greeting[1:]
         print greeting
         print new_greeting
```

```
Hello, world!
Jello, world!
```

7 8.6 Searching

```
In [9]: # What does the following FUNCTION do?
        # Compare with the METHOD 'find' from the 8.8 paragraph!
        def find (word, letter):
            index = 0
            while index < len(word):
                if word[index] == letter:
```

```
                return index
            index = index + 1
        return -1
    """
    Answer: this function is the opposite of the [] operator.
    Instead of taking an index and
    extracting the corresponding character, it takes a character
    and finds the index where that character appears.
    If the character is not found, the function returns -1.
    """
    find ('abracadabra','d')
    # What about finding 'a' in 'abracadabra'? Or 'b'?
```

Out[9]: 6

```
In [ ]: """
        SEARCHING is the pattern of computation based on traversing
        a sequence and returning when
        we find what we are looking for.
        """
```

7.0.4 Exercise 8.4. (page 74)

Modify the function "find()" so that it has a third parameter, the index in "word" where it should start looking.

```
In [ ]: def findFrom():
            ...
```

8 8.7 Looping and counting

```
In [28]: # The following program counts the number of times the letter
         # 'a' appears in a string.
         word = 'banana'
         count = 0
         for letter in word:
             if letter == 'a':
                 count = count + 1
         print count
```

```
3
```

```
In [ ]: """
        COUNTING is a pattern of computation based on initializing
        a counter variable to 0 and then incrementing it
        each time the condition is valid.
        """
```

8.0.5 Exercise 8.5. (page 75)

Encapsulate the previous code in a function named "count", and generalize it so that it accepts the string and the letter as arguments.

```
In [ ]: def count(...):
            ...
```

8.0.6 Exercise 8.6. (page 75)

Rewrite this function (from the previous exercise) so that instead of traversing the string, it uses the three-parameter version of "find" from the previous section.

```
In [ ]: def count3param(...):
            ...
```

9 8.8 String methods

```
In [29]: """
         A METHOD is similar to a function BUT the syntax is different.
         """
         word = 'banana'
         new_word = upper(word)
         print new_word


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-29-d86935517d8f> in <module>()
          3 """
          4 word = 'banana'
    ----> 5 new_word = upper(word)
          6 print new_word


        NameError: name 'upper' is not defined
```

```
In [30]: word = 'banana'
         new_word = word.upper()
         print new_word
```

```
BANANA
```

```
In [ ]: """
        This form of dot notation specifies the name of the method, upper,
        and the name of the object - the string to apply the method to, word.
        """
```

```
In [31]: word = 'banana'
         index = word.find('a')
         print index
```

```
1
```

```
In [32]: word = 'banana'
         index = word.find('na')
         print index
```

```
2
```

```
In [ ]: # Compare this METHOD 'find' with the FUNCTION 'find' from the 8.6 paragraph!
```

9.0.7 Exercise 8.7. (page 76)

There is a string method called "count" that is similar to the function in the previous exercise. Read the documentation of this methid and write an invocation that counts the numebr of 'a's in "banana".

```
In [ ]: #invoke the in-built count method
```

9.0.8 Exercise 8.8. (page 76)

Read the documentation of the "string" methods at http://docs.python.org Useful methods: strip, replace, etc.

10 8.9 The in operator

```
In [33]: # The operator "in" is a boolean operator that takes two strings and
         # returns True if the first appears as a substring in the second.
         print 'a' in 'banana'
```

```
True
```

```
In [35]: def in_both(word1, word2):
             for letter in word1:
                 if letter in word2:
                     print letter

         in_both('apples', 'oranges')
```

```
a
e
s
```

11 8.10 String comparison

```
In [10]: # The relational operators work on strings.
         def compare_strings(word1, word2):
             if word1 == word2:
                 print 'All right, ', word1, 's!'
             elif word1 < word2:
                 print 'Your word', word1, 'comes before', word2, '. '
             else:
                 print 'Your word', word1, 'comes after', word2, '. '

         w1 = raw_input('Your first word is: ')
         w2 = raw_input('Your second word is: ')
         compare_strings(w1, w2)
         # All the uppercase letters come BEFORE all the lowercase letters.
```

```
Your first word is: w
Your second word is: an
Your word w comes after an .
```

11.0.9 Exercise 8.9. (page 78)

Starting with the diagram from the Debugging section, execute the program on paper, changing the values of i and j during each iteration. Find and fix the second error in this function...

12 Debugging, Glossary, Exercises

12.0.10 Exercise 8.10. (page 79)

12.0.11 Exercise 8.11 (page 79)

12.0.12 Exercise 8.12 (page 80)

... rotate.py