_C7-2_90_ThinkPython-Chap10-Theory

April 10, 2020

1 Lists

2 10.1 A list is a sequence

```
In [ ]: # 10.1. A list is a sequence of values.
        # In a string - the values are characters.
        # In a list - the values can be any type.

        # The values in a list are called elements or items.
```

```
In [54]: # Examples
         [10, 20, 30, 40]
         ['string1', 'string1 string2', ' ']

         myList = ['spam', 2.0, 5, [10, 20]] #nested list
         print "Lista ", myList, "este cu elemente de tipuri diferite. "

         l1 = [] #empty list
         l2 = [''] #non-empty list
         l3 = ['\n', '/n']
         print "Liste speciale sunt:", l1, l2, '\n', '/n', l3
```

```
Lista  ['spam', 2.0, 5, [10, 20]] este cu elemente de tipuri diferite.
Liste speciale sunt: [] ['']
/n ['\n', '/n']
```

```
In [55]: print "len(myList) = ",len(myList)
         print "len(l1) = ", len(l1)
         print "len(l2) = ", len(l2)
```

```
len(myList) =  4
len(l1) =  0
len(l2) =  1
```

3 10.2 Lists are mutable

```
In [56]: # The expression inside the brackets specifies the index.
         # The indices start from 0.
         list0 = ['spam', 2.0, 5, [10, 20]]
         print list0[0]

         # You can think of a list as a relationship between indices and elements.
         # This relationship is called mapping.
```

```
spam
```

```
In [57]: # UNLIKE STRINGS, lists are mutable!!! :)
         list1 = ['spam', 2.0, 5, [10, 20]]
         list1[1] = 20
         print list1
```

```
['spam', 20, 5, [10, 20]]
```

```
In [58]: """
         List indices work the same way as string indices:
          - any integer expression could be used as an index;
          - if you try to read or write an element that does not exist,
            you get an IndexError;
          - IF AN INDEX HAS A NEGATIVE VALUE,
            IT COUNTS BACKWARD FROM THE END OF THE LIST, starting with -1.
         """
         print list1[-3]
```

```
20
```

```
In [13]: # The in operator also works on lists

         lst = ['spam', 20, 5, [10, 20]]
         print "20 in lst --> ", 20 in lst
         print "10 in lst --> ", 10 in lst

         lst[3] = [10,5]
         print "lista noua este: ", lst

         print 'Surpriza :) ', lst[3][0]

         lst[3] = 10
         print "10 in lst --> ", 10 in lst
```

```
20 in lst -->  True
10 in lst -->  False
```

```
lista noua este:  ['spam', 20, 5, [10, 5]]
Surpriza :)  10
10 in lst -->  True
```

4 10.3 Traversing a list

```
In [1]: # Use the elements only for reading them
        listList = [[1],[2,5,3], [10,4,8,4]]
        for l in listList:
            print 10 in l
```

```
False
False
True
```

```
In [25]: # To use the elements for writing or update them, you need the INDICES
         numbers = [10, 30, 20, 50]
         for i in range(len(numbers)):
             numbers[i] += 10
         print numbers
```

```
[20, 40, 30, 60]
```

```
In [2]: # A for loop over an empty list never executes the body.
        for x in []:
            print "Nu se executa!"
        print ":)"
```

```
:)
```

5 10.4 List operations

```
In [27]: # The operator + concatenates the lists
         print [1,3,5] + [2,4,6]
```

```
[1, 3, 5, 2, 4, 6]
```

```
In [3]: # The * operator repeats a list a given number of times
        print [0] * 4
        print 4 * [[0]]
        print [1,2] * 2
```

```
[0, 0, 0, 0]
[[0], [0], [0], [0]]
[1, 2, 1, 2]
```

6 10.5 List slices

```
In [33]: # The slice operator works on lists as it works on strings.
         chars = ['a', 'b', 'c', 'd', 'e', 'f']
         print chars[1:3]
         print chars[:4]
         print chars[3:]

         Chars = chars[:]
         print Chars + ['g']
```

```
['b', 'c']
['a', 'b', 'c', 'd']
['d', 'e', 'f']
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

```
In [7]: # A slice operator on the left side of an assignment can
        # UPDATE MULTIPLE elements
        t = ['1', '2', '3', '4', '5']
        t[1:3] = ['x', 'y', 'z', 1]
        print t
```

```
['1', 'x', 'y', 'z', 1, '4', '5']
```

```
In [8]: t = ['1', '2', '3', '4', '5']
        t[1:3] = ['x', 'y']
        t[1:3] *= 2
        print t
```

```
['1', 'x', 'y', 'x', 'y', '4', '5']
```

7 10.6 List methods

```
In [ ]: # Python provides MMEETTHHOODDSS that operates on lists.
```

```
In [9]: l = ['a', 'd', 'f']
        l.append('b')
        print l
        #l.append('x', 'y')
        #print l
        l.append([1,2])
        print l
```

```
['a', 'd', 'f', 'b']
['a', 'd', 'f', 'b', [1, 2]]
```

```
In [41]: l.extend(['c', 'e'])
         print l

['a', 'd', 'f', 'b', 'c', 'e']


In [24]: l1 = ['a', 'e', 'c', 'b', 'd', 'f']
         l1.sort()
         print 'lista1 = ', l1

         l2 = [1, 'a', '2', 3, False, 'aa']
         l2.sort()
         print 'lista2 = ', l2

lista1 =  ['a', 'b', 'c', 'd', 'e', 'f']
lista2 =  [False, 1, 3, '2', 'a', 'aa']


In [17]: """
         !!!!!MOST OF THE LIST METHODS ARE void!!!!!
         Exception: pop method --> see below
         They modify the list and return None.
         """
         test = [1, 3, 2, 7, 4]
         test = test.sort()
         print test

None
```

8 10.7 Reduce, map, filter and list comprehension

```
In [11]: # An operation that combines a sequence of elements into
         # a single values is sometimes called reduce.
         nr = [1, 2.8, 3]
         print sum(nr)

         print '1' + '2' + '3'

         chars = ['1', '2', '3']
         print sum(chars)

6.8
123
```

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

```
    TypeError                                 Traceback (most recent call last)

    <ipython-input-11-f9df3140a719> in <module>()
      6 print '1' + '2' + '3'
      7 chars = ['1', '2', '3']
----> 8 print sum(chars)


    TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

8.0.1 Exercise 10.1. (page 91)

Write a function called "nested_sum" that takes a nested list of integers and add up the elements from all of the nested lists.

```
In [ ]: def nested_sum(intListList):
            ...
```

```
In [20]: # An operation is called a map if it maps a function onto
         # each of the elements in a sequence.
         def cap_all(lst):
             cLst = []; ccLst = []
             for e in lst:
                 cLst.append(e.capitalize())
                 ccLst.append(e.upper())
             return cLst, ccLst

         print cap_all(['ana', 'Baba', '1a'])

         # Using { } instead of [ ] we have SET instead of LIST
         # The data structure SET represents the mathematical set, meaning
         # no repetitions, no elements order
         print cap_all({'ana', 'Baba', '1a'})
         # :)))) ?!?!
         """
         Python takes the liberty to change the order
         of the elements for the needs of the internal
         implementation it uses for a set
         """
```

```
(['Ana', 'Baba', '1a'], ['ANA', 'BABA', '1A'])
(['Baba', '1a', 'Ana'], ['BABA', '1A', 'ANA'])
```

```
In [19]: people = {"Jay", "Idrish", "Archil"}
         vampires = {"Karan", "Arjun"}
         population = people.union(vampires)
         print population
```

```
set(['Arjun', 'Archil', 'Idrish', 'Karan', 'Jay'])
```

8.0.2 Exercise 10.2. (page 91)

Use "capitalize_all" to write a function named "capitalize_nested" that takes a nested list of strings and returns a new nested list with all strings capitalized.

```
In [ ]: def capitalize_nested(strListList):
            ...
```

```
In [18]: # Another common operation is TO SELECT or FILTER some of the elements
         # from a list and return a sublist.
         def only_upper(lst):
             uLst = []
             for e in lst:
                 if e.isupper():
                     uLst.append(e)
             return uLst

         print only_upper(['a', 'B', 'c'])
```

```
['B']
```

```
In [60]: # There is a built-in function "map" in Python.
         # There is an operator "list comprehension" in Python.
         test = 'hello'
         parts = [c for c in test]
         print parts
```

```
['h', 'e', 'l', 'l', 'o']
```

8.0.3 Exercise 10.3. (page 92)

Write a function that takes a list of numbers and returns the cumulative sum; that is, a new list where the element with the index "i" is the sum of the first i+1 elements from the original list. For example, the cumulative sum of [1,2,3] is [1,3,6]

```
In [ ]: def cumSum(numList):
            ...
```

9 10.8 Deleting elements

```
In [50]: # The pop method deletes and EXTRACTS an element given by its index.
         # So, the pop method is fruitful (it is not a void method).
         nr = [1, 3, 2, 7, 4]
         elem = nr.pop(3)
         print "Lista devine: ", nr
         print elem
         # If you don't provide an index as argument,
         # it deletes and returns THE LAST element.
         elem = nr.pop()
         print "Lista devine: ", nr
         print elem
```

```
Lista devine:  [1, 3, 2, 4]
7
Lista devine:  [1, 3, 2]
4
```

```
In [51]: # If you don't need the removed value you may use the del OOPPEERRAATTOORR.
         nr = [1, 3, 2, 7, 4]
         del nr[3]
         print nr
```

```
[1, 3, 2, 4]
```

```
In [10]: # If you know the value of the element you want to remove
         # (but not the index), you can use remove method.
         nr = [1, 3, 2, 7, 4]
         nr.remove(3)
         print nr
         # The return value from remove is None.
         nr = [1, 3, 2, 3, 4]
         nr.remove(3)
         print nr
```

```
[1, 2, 7, 4]
[1, 2, 3, 4]
```

```
In [53]: # In order to remove more than one element,
         # you can use del with a slice index.
         c = ['a', 'b', 'c', 'd', 'e', 'f']
         del c[2:5]
         print c
```

```
['a', 'b', 'f']
```

9.0.4 Exercise 10.4. (page 92)

Write a function called “middle” that takes a list and returns a new list that contains all but the first and the last elements. So, middle([1,2,3,4])=[2,3]

```
In [ ]: def middle(myList):
            ...
```

9.0.5 Exercise 10.5. (page 92)

Write a function called “chop” that takes a list, modifies it by removing the first and the last elements, and returns the special value None.

```
In [ ]: def chop(myList):
            ...
```

10 10.9 Lists and strings

```
In [26]: """Strings, lists, and tuples are objects,
         which means that they not only hold values,
         but have built-in behaviours called methods,
         that act on the values in the object.
         """

         # A list of characters IS NOT THE SAME as a string.
         # A string is a list of characters in order.

         sirLista = ['1','2','3']
         sir = '123'

         print len(sirLista), len(sir)

         print sirLista[1:3], sir[1:3]

         sirLista[0] = '4'
         print sirLista # Lists are mutable

         sir[0] = '4'
         print sir # Strings are immutable
```

```
3 3
['2', '3'] 23
['4', '2', '3']



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

        TypeError                                 Traceback (most recent call last)

        <ipython-input-26-d05c344424ff> in <module>()
         18 print sirLista
         19
    ---> 20 sir[0] = '4'
         21 print sir
    TypeError: 'str' object does not support item assignment
```

```
In [21]: # To convert from a string to a list of characters,
         # you can use the BUILT-IN FUNCTION list.
         print list('spam')
         print list(['s', 'p', 'a', 'm'])
```

```
['s', 'p', 'a', 'm']
['s', 'p', 'a', 'm']
```

```
In [48]: # If you want to break a string into words, you can use the split METHOD.
         phrase = 'Ana are mere.'
         words = phrase.split()
         print words
         # The default delimiter is the space.
```

```
['Ana', 'are', 'mere.']
```

```
In [68]: # The argument of the split method could be the delimiter.
         phrase = 'Ana are mere, pere, nuci.'
         words = phrase.split()
         print words
         fruits = phrase.split(',')
         print fruits
```

```
['Ana', 'are', 'mere,', 'pere,', 'nuci.']
['Ana are mere', ' pere', ' nuci.']
```

```
In [70]: # The inverse METHOD of split is join.
         # join is a string method.
         t = ['Ana', 'are', 'mere.']
         delimiter = ' '
         print delimiter.join(t)
         delimiter = '-'
         print delimiter.join(t)
```

```
Ana are mere.
Ana-are-mere.
```

11 10.10 Objects and values

```
In [71]: # For the following code, what about the values for a and b?
         a = 'banana'
         b = 'banana'
         # Do the variable a and b refer to THE SAME string?
         # We may check this using the "is" operator.
         print a is b # ONE STRING OBJECT and both a and b refer to it!!!
         # The string value unifies the objects!
```

```
True
```

```
In [72]: # BUT, when you create TWO LISTS, YOU GET TWO OBJECTS!!!
         a = ['ba', 'na', 'na']
         b = ['ba', 'na', 'na']
         print a is b
         # We call these lists as being equivalent -
         # they have the same elements, but they are not identical -
         # because they are not the same object.
```

```
False
```

12 10.11 Aliasing

```
In [4]: # If "a" refers to an object and you assign b = a,
        # THEN BOTH variables refer to the same object.
        a = [1,2,3]
        b = a
        print 'pt. lista', b is a

        c = 'banana'
        d = c
        print 'pt. string', d is c
```

```
pt. lista True
pt. string True
```

```
In [36]: # The association of a variable with an object
         # is called a REFERENCE.

         # An object with more than one reference has
         # more than one name and
         # we may say that the object is aliased.

         # If the aliased object is mutable,
         # CHANGES MADE with one alias affect the other.

         a = [1,2,3]
         b = a
```

```
        b[0] = 17
        print 'a =', a

        c = 'banana'
        d = 'banana'
        # d[0] = 'B' ERR, strings are immutable!!!
        d = 'B' + 'anana'
        print 'd = ', d
        print 'c = ', c
```

```
a = [17, 2, 3]
d =  Banana
c =  banana
```

```
In [ ]: # Although this behaviour can be useful, it is error-prone.
        # IN GENERAL, IT IS SAFER to avoid aliasing
        # WHEN you are working with mutable objects.
```

13 10.12 List arguments

```
In [48]: # When you pass a list to a function, the function gets
         # a REFERENCE to that list.
         # If the function modifies the list parameter,
         # the caller sees the change.

         # WRONG example:
         def bad_delete_head(lst):
             lst = lst[1:]
             # The slice operator creates a new list and
             # the assignment makes lst TO REFER TO that new list,
             # BUUUUT NONE OF THAT has any effect on the original list
             # that was passed as as an argument :(((
             print 'Lista in functia BAD', lst

         # Correct examples
         def delete1_head(lst):
             del lst[0]

         def delete2_head(lst):
             return lst[1:] # nu modifica, returneaza rezultatul unei expresii

         lst = ['1','2','3']
         bad_delete_head(lst)
         print 'Lista dupa apel BAD este: ', lst

         delete1_head(lst)
         print "Lista returnata dupa del este: ", lst
```

```
        print "Lista returnata cu return...slice este: ", delete2_head(lst)

        print "Lista finala este: ", lst
```

```
Lista in functia BAD ['2', '3']
Lista dupa apel BAD este:  ['1', '2', '3']
Lista returnata dupa del este:  ['2', '3']
Lista returnata cu return...slice este:  ['3']
Lista finala este:  ['2', '3']
```

14 Debugging, Glossary, Exercises

14.0.6 Exercise 10.6. (page 98)

Write a function called "is_sorted" that takes a list as a parameter and returns True if the list is sorted in ascending order and False otherwise. You can assume (as a precondition) that the elements of the list can be compared with the relational operators <, >, etc.

```
In [ ]: def is_sorted(myList):
            ...
```

14.0.7 Exercise 10.7. (page 98)

Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called "is_anagram" that takes two strings and returns True if they are anagrams.

```
In [ ]: def is_anagram(w1,w2):
            ...
```

14.0.8 Exercise 10.8. (page 98)

… birthday.py…

14.0.9 Exercise 10.9. (page 98)

Write a function called "remove_duplicates" that takes a list and returns a new list with only the unique elements frm the original. Hint: they don't have to be in the same order.

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

14.0.10 Exercise 10.10. (page 98)

… wordlist.py

14.0.11 Exercise 10.11. (page 98)

… inlist.py

14.0.12 Exercise 10.12. (page 99)

… reverse_pair.py

14.0.13 Exercise 10.13. (page 99)

… interlock.py … http://puzzlers.org