Python Data Structures: Lists:

Programming for Data Science with Python

1. Overview

In Python, ***lists*** are the objects of the class list that has the ***constructor list()***.

A list is a ***mutable*** sequence data type/structure, i.e., its ***contents can be changed*** after being created.

List literals are written within square brackets [ ].

Lists work similarly to strings:

- Use the len() function for the length of a list
- Use square brackets [ ] to access data, with the first element at index 0
- The range of indices: 0 .. len(a list) - 1

1.1 Properties of Lists

The ***main properties*** of Python lists:

- List elements are ordered in a sequence.
- List contain objects of different data types
- Elements of a list can be accessed by an index - as other sequence data type/structures like strings, tuples
- Lists are arbitrarily nestable, i.e. they can contain other lists as sublists
- Lists are ***mutable***, i.e. their elements can be changed after the list has been created.

Examples:

Empty list

```
my_list=[]
```

List of integers

```
my_list = [1,2,3]
```

List with mixed datatypes

```
my_list = [1, "Hello", 3.4]
```

Nested list

> my_list =["mouse", [8,4,6], ['a']]

1.2. Elements of a list

Index range of list elements

Forward index range of list elements: ***0 .. len(list) - 1*** Forward: starting from the 1st element

Backward index range of list elements: -1 .. -len(list) Backward : Starting from the last element

1.3. Constructor list(iterable)

The ***constructor list()*** builds a list whose items are the same and in the same order as iterable's items.

- ***iterable*** may be either a sequence, a container that supports iteration, or an iterator object.
- If ***iterable is already a list, a copy*** is made and returned, similar to iterable[:].

For example:

- list('abc') returns ['a', 'b', 'c']
- list((1, 2, 3)) returns [1, 2, 3].

If ***no argument*** is given, the constructor creates a ***new empty list, []***.

Run the following 3 code blocks:

In [4]:

```python
list("abc")
```

Out[4]:

```
['a', 'b', 'c']
```

In [5]:

```python
list ((1,2,3))
```

Out[5]:

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

In [6]:

```python
list ([1, 3, 5, 7, 9])
```

Out[6]:

```
[1, 3, 5, 7, 9]
```

2. Create Lists

2.1 Overview

Lists may be constructed in several ways:

- Using a pair of square brackets to denote the ***empty list: []***
- Using square brackets with values separating from each others with commas: [a], [a, b, c]
- Using a ***list comprehension:*** [x for x in iterable]
- Using the ***list constructor:*** list() or list(iterable)

2.2 Create empty lists

Run the following code block:

In [8]:

```python
empty_list = []
another_empty_list = list()
print(len(empty_list))
print(len(another_empty_list))
```

```
0
0
```

2.3 Create lists by converting other data structures/types to lists: Using list()

2.3.1 Create list from strings or tuples using the constructor list()

Run the following 3 code blocks:

In [9]:

```python
# Convert a string of one word to a list of characters
list("house")

```

Out[9]:

```
['h', 'o', 'u', 's', 'e']
```

In [10]:

```python
# Convert a a string of words to a list of characters
list("This word")
```

Out[10]:

```
['T', 'h', 'i', 's', ' ', 'w', 'o', 'r', 'd']
```

In [13]:

```python
# Convert a tuple of a list
# Notice the parentheses vs. the square brackets
aTuple = ('ready', 'fire', 'aim')
list(aTuple)
```

Out[13]:

```
['ready', 'fire', 'aim']
```

2.3.2 Create lists from strings using split() method

Run the following 2 code blocks:

In [16]:

```python
#Convert a string of words to a list of words: Using split() to chop the string with '

aStringOfWords= "This is a string of words"
aList=aStringOfWords.split(' ')
print(aList)
```

```
['This', 'is', 'a', 'string', 'of', 'words']
```

In [17]:

```python
#Convert a string to a List: Using split() to chop the string with some separator
aDayString = "5/1/2017"
alist = aDayString.split('/')
print(alist)
```

```
['5', '1', '2017']
```

2.3.3 Create lists by using list comprehension and slicing an existing list

Run the following code block:

In [18]:

```python
# NOTES: MUST use List slice--> CANNOT use any other function to delete/remove

l_lists=[[1,2,3],[2,3,4],[3,4,5]]

new_llists=[element[1:] for element in l_lists]

i=0
for element in new_llists:
    print(element)
    i=i+1
    if i==3:
        break
```

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

3. Access List Elements

3.1 Access single elements

- As other sequence data types/structures, list elements can be accessed via their indices.
- We can use the index operator [] to access an item in a list. ***Index starts from 0.***
- So, a list having 5 elements will have index from O to 4.
- Trying to access an element other than this will raise an IndexError.
- ***The index must be an integer.***
- We can't use float or other types, this will result into TypeError.

Nested list are accessed using ***nested indexing [][]*** that is similar to index of 2-D array elements.

Run the following 6 code blocks:

In [19]:

```python
my_list = ['p','r','o','b','e']


print(my_list[0])

print(my_list[2])

print(my_list[4])

```

```
p
o
e
```

In [20]:

```
# Nested List

n_list = ["Happy", [2,0,1,5]]

# Nested indexing

print(n_list[0][1])

print(n_list[1][3])
```

```
a
5
```

In [21]:

```
aTuple=('ready','fire','aim')
aList=list(aTuple)

print (aList)
print("Length of the list:",len(aList))
```

```
['ready', 'fire', 'aim']
Length of the list: 3
```

In [22]:

```
# Access using forward index

aTuple=('ready','fire','aim')
aList=list(aTuple)

list_element1=aList[0]
list_element2=aList[1]
list_element3=aList[2]

print(list_element1)
print(list_element2)
print(list_element3)
```

```
ready
fire
aim
```

In [23]:

```
# Access using backward index
aTuple=('ready','fire','aim')
aList=list(aTuple)

list_element_last=aList[-1]
list_element_next_to_last=aList[-2]
list_element_first=aList[-3]

print(list_element_last)
print(list_element_next_to_last)
print(list_element_first)
```

```
aim
fire
ready
```

In [24]:

```
languages= ["Python", "C", "C++", "Java", "Perl"]
print(languages[0] +" and "+ languages[1] +" are quite different!")
```

```
Python and C are quite different!
```

3.2 Access a slice of lists

Run the following code block:

In [25]:

```
# We can access a range of items in a List by using the slicing operator (colon).
# This is a very important concept for when we start working with algorithms in the 2nd

my_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd up to the 5th (but not including)
print(my_list[2:5])

# elements backward from (but not inclucing) the negative 5th element ("r")
print(my_list[:-5])

# elements 6th to end
# Remember the count starts at zero, not one
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

```
['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

4. Modify Lists

4.1 Add/Change elements of lists

> **4.1.1 Update/Change single elements or a sub-list of lists**

Run the following code block:

In [26]:

```
odd= [2, 4, 6, 8]

# change the 1st item
odd[0] = 1
print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]
print(odd)
```

```
[1, 4, 6, 8]
[1, 3, 5, 7]
```

> **4.1.2 Add single items or a sub-list into a list - using append() or extend() respectively**

Run the following code block:

In [27]:

```
# We can add one item to a List using append() method
# or add several items using extend() method.
odd= [1, 3, 5]

odd.append(7)
print(odd)

odd . extend([9, 11, 13])
print(odd)
```

```
[1, 3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
```

> **4.1.3 Insert single elements or sub-lists into an existing list**

Run the following code block:

In [28]:

```python
# We can insert one item at a desired Location by using the method insert()
# or insert multiple items by squeezing it into an empty slice of a List.

odd= [1, 9]
odd.insert( 1,3)
print(odd)

odd[2:2] = [5, 7]
print(odd)
```

```
[1, 3, 9]
[1, 3, 5, 7, 9]
```

4.2 Delete/Remove elements of lists

4.2.1 Delete/Remove elements of lists - using the del() function

Run the following code block:

In [29]:

```python
# We can delete one or more items from a List using the keyword del.

my_list = ["p","r", "o", "b", "l", "e", "m"]

# delete one item
del my_list[2]
print("3rd element has been removed: ", my_list)

# delete multiple items
del my_list[1:5]
print("Elements from index 1 until 4 have been removed: ", my_list)
```

```
3rd element has been removed:  ['p', 'r', 'b', 'l', 'e', 'm']
Elements from index 1 until 4 have been removed:  ['p', 'm']
```

4.2.2 Delete/Remove elements of lists - using the functions remove() or pop{)

Run the following code block:

In [30]:

```
# We can use remove() method to remove the given item or pop() method to remove an item
# The pop() method removes and returns the Last item if index is not provided.
# This helps us implement lists as stacks (first in, Last out data structure).
# We can also use the clear() method to empty a List.

my_list=['p','r','o','b','l','e','m']

# Remove p, p is gone. ("r", "o", "b", "l", "e", "m") is left.
my_list.remove('p')

# Will now remove the first element ("o"). ("r","b","l","e","m") is left.
my_list.pop(1)

# Will now remove the last element
my_list.pop()

print(my_list)
```

```
['r', 'b', 'l', 'e']
```

4.2.3 Delete/Remove elements of a list - assigning an empty list [] to a slice of the list

Run the following code block:

In [31]:

```
my_list=['p','r','o','b','l','e','m']

# remove 'o'
my_list[2:3]=[]

# remove 'b', 'l', 'e'
my_list[2:5]=[]

print(my_list)
```

```
['p', 'r', 'm']
```

4.2.4 Delete/Remove all the elements of a list - using the clear() function

Run the following code block:

In [32]:

```
my_list=['p','r','o','b','l','e','m']
my_list.clear()

print(my_list)
```

```
[]
```

5. Copy Lists

5.1 Shallow copy

- ***Shallow copy*** means that only the reference to the object is copied. No new object is created.
- ***Shallow Copy*** means defining a new collection object and then populating it with references to the child objects found in the original.
- The ***Shallow Copy*** process is not recursive. This means that the child objects won't be copied. In case of shallow copy, a reference of object is copied in other object. It means that any changes made to a copy of object do reflect in the original object. In python, this is implemented using “copy()” function.

Run the following code block:

In [35]:

```python

# importing "copy" for copy operations
import copy

# initializing list 1
i1 = [1, 2, [3,5], 4]

# using copy to shallow copy
s2 = copy.copy(i1)

# original elements of list
print ("The original elements before shallow copying")
for i in range(0,len(i1)):
    print (i1[i],end=" ")

print("\n")

# modifying the new list (shallow copy)
s2[2][0] = 7

# checking if change is reflected
print ("The original elements after shallow copying")
for i in range(0,len( i1)):
    print (i1[i],end=" ")

```

```
The original elements before shallow copying
1 2 [3, 5] 4

The original elements after shallow copying
1 2 [7, 5] 4
```

5.2 Deep copy

- The ***Deep Copy*** process is where the copying process occurs recursively.
- ***Deep copy*** means a new collection will first be created and then that copy will recursively be populated with copies of the child objects found in the original list.
- A ***Deep Copy*** stores copies of an object's values, but a ***Shallow Copy*** stores references to the original object(list, dict, etc)
- A ***Deep Copy** does **_NOT_** reflect any changes made to the new (copied) object from the original object; however, the ***Shallow Copy*** does reflect any modifications.
- A ***Deep Copy*** is the ***real copy*** of the orginal.
- Deep copying lists can be done using the ***deepcopy()*** function of the ***module copy*** in Python 3.

Run the following code block:

In [36]:

```
# importing "copy" for copy operations
import copy

# initializing list 1
i1 = [1, 2, [3,5], 4]

# using deepcopy() to deep copy initial list (il)
d2 = copy.deepcopy(i1)

# original elements of list
print ("The original elements before deep copying")
for i in range(0,len(i1)):
    print (i1[i],end=" ")

print("\n")

# adding and element to new list
d2[2][0] = 7

# Change is reflected in l2
print ("The new list of elements after deep copying ")
for i in range(0,len( i1)):
    print (d2[i],end=" ")

print("\n")

# Change is NOT reflected in original list
# as it is a deep copy
print ("The original elements after deep copying")
for i in range(0,len( i1)):
    print (i1[i],end=" ")

```

```
The original elements before deep copying
1 2 [3, 5] 4

The new list of elements after deep copying
1 2 [7, 5] 4

The original elements after deep copying
1 2 [3, 5] 4
```

6. Delete Lists

To delete a list, using the built-in function del().

Run the following 3 code blocks:

In [37]:

```python
list1 = [1, 2, [3,5], 4]
print(list1)
```

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

In [38]:

```python
del(list1)
print("list1 has been deleted.")
```

```
list1 has been deleted.
```

In [39]:

```python
print(list1)
# You will get an error since list1 has been deleted.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-39-476b7d506017> in <module>
----> 1 print(list1)
      2 # You will get an error since list1 has been deleted.

NameError: name 'list1' is not defined
```

7. Operations on List

Lists implement all of the common and mutable sequence operations.

7.1 Concatenate lists

Using + to concatenate strings

Run the following 2 code blocks:

In [40]:

```python
list1 = [1, 2, [3,5], 4]
list2 = ["Hello", "World"]
print(list1 + list2)
```

```
[1, 2, [3, 5], 4, 'Hello', 'World']
```

In [42]:

```
# We can also use+ operator to combine two lists.
#This is also called concatenation.
#The * operator repeats a list for the given number of times.

odd= [1, 3, 5]


print(odd + [9, 7, 5])
```

```
[1, 3, 5, 9, 7, 5]
```

7.2 Replicate lists

Run the following 2 code blocks:

In [43]:

```
aList = [1, 2]

print (aList * 3)
```

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

In [44]:

```
print(["re"] * 3)
```

```
['re', 're', 're']
```

7.3 Test elements with "in" and "not in"

Run the following 2 code blocks:

In [45]:

```
list1 = [1, 2, [3,5], 4]
print (2 in list1)
```

```
True
```

In [46]:

```
list1 = [1, 2, [3,5], 4]
print ([3] in list1)
```

```
False
```

7.4 Compare lists: <, >, <=, >=, ==, !=

Run the following code block:

In [47]:

```
list1 = [1, 2, [3,5], 4]
list2 = [1, 2, 4]
print (list1 == list2)
```

```
False
```

7.5 Iterate a list using for loop

Run the following 4 code blocks:

In [48]:

```
list1 = [1, 2, [3,5], 4]
for i in list1:
    print (i)
```

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

In [61]:

```
list1 = [1, 2, [3,5], 4]

for i in list1:
    print(i, end="")
```

```
12[3, 5]4
```

In [50]:

```
list1 = [1, 2, [3,5], 4]
for i in list1:
    print (i, end="\n")
```

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

In [60]:

```
for fruit in ["apple","banana","mango"]:
    print("I like",fruit)
```

```
I like apple
I like banana
I like mango
```

7.6 Sort lists

7.6.1 Using the sort method of the class list: sort (*, key = none, reverse = false)

This method list.sort():

- Sort the list in ***place***
- Use only < comparisons between items.

By default, sort() doesn't require any extra parameters . However, it has two optional parameters :

- reverse - If true, the sorted list is reversed (or sorted in descending order)
- key - function that serves as a key for the sort comparison

IMPORTANT NOTES:

This method modifies the sequence in place for economy of space when sorting a large sequence. Exceptions are not suppressed.

- if any comparison opertions fail, the entire sort operation will fail
- the list will likely be left in a partially modified state.

Run the following code block:

In [59]:

```python
# vowels list
vowels= ['e', 'a', 'u', 'o', 'i']

# sort the vowels
vowels.sort()

# print vowels
print('Sorted list:', vowels)
```

```
Sorted list: ['a', 'e', 'i', 'o', 'u']
```

7.6.2 Using the built-in sorted() function: sorted(iterable, *, key = None, reverse = False)

The built-in sorted() function returns a new sorted list from the items in iterable.

Run the following code block:

In [58]:

```
# vowels list
vowels= ['e', 'a', 'u', 'o', 'i']

# sort the vowels
sortedVowels = sorted(vowels)

# print vowels
print('Sorted list:', sortedVowels)

#A new list has been created and returned by the built-in sorted function
id(vowels), id(sortedVowels)
```

```
Sorted list: ['a', 'e', 'i', 'o', 'u']
```

Out[58]:

```
(2102530710400, 2102530710336)
```

8. Class list

7.1 Count()

count(x): return the number of elements of the tuple that are equal to x

Run the following code block:

In [57]:

```
list1 = ['a','p','p','l','e']
print(list1.count('p'))
```

```
2
```

7.2 index (x)

index(x) returns the index of the first element that is equal to x

Run the following code block:

In [56]:

```
list1 = ['a','p','p','l','e']
print(list1.index('p'))
```

```
1
```