Python Lists Notes

Shubham Verma

Linkedin https://www.linkedin.com/in/shubham-verma-3968a5119

Note: Please follow my Linkedin for Question on Lists

1.0 List introduction

1.1 Syntax

1. Lists are used to store multiple items in a single variable.
2. Lists are created using square brackets
3. Lists can store multiple data types also for ex. tuple, sets, dict, bool, etc

for example see below list, l

```
In [52]: l = [1,2,"shubham", [1,23,4], (1,2,3), {1,2}, {"key": "value"}, True]
         l
Out[52]: [1, 2, 'shubham', [1, 23, 4], (1, 2, 3), {1, 2}, {'key': 'value'}, True]
```

1.2 type() and len() Method

1. type() function is used to get data type
2. len() function is used to get length of list

```
In [53]: type(l)
Out[53]: list
```

```
In [54]: len(l)
Out[54]: 8
```

1.3 List Properties

1. lists are ordered, it means that if you add new items to a list, the new items will be placed at the end of the list. One exception to this is insert() method which inserts element at specified index.
2. list is mutable, it means that we can change, add, and remove items in a list after it has been created.
3. lists allow duplicates, since lists are indexed, so lists can have items with the same value.

1.4 list() Method

1. list() is used to convert other data types to list

```
In [55]: s = "Sameer"
         list(s)
Out[55]: ['S', 'a', 'm', 'e', 'e', 'r']
```

```
In [56]: tuple1 = (1,2,3,4,5,6)
         list(tuple1)
Out[56]: [1, 2, 3, 4, 5, 6]
```

2.0 Accessing list elements

1. list can be accessed by positive and negative indexing.

```
In [57]: list1 = ["Red", "Blue", "Green","Yellow", True, False, 1,3]
```

2.1 Positive indexing

```
In [58]: # from first to last element
         list1[0:]

Out[58]: ['Red', 'Blue', 'Green', 'Yellow', True, False, 1, 3]
```

```
In [59]: # from first to nth element
         list1[:4]

Out[59]: ['Red', 'Blue', 'Green', 'Yellow']
```

```
In [60]: # range of element
         list1[3:6]

Out[60]: ['Yellow', True, False]
```

```
In [61]: # elements with a jump 2 by default jump is 1
         list1[0::2]

Out[61]: ['Red', 'Green', True, 1]
```

2.2 Negative indexing

```
In [62]: # upto second last element
         list1[:-1]

Out[62]: ['Red', 'Blue', 'Green', 'Yellow', True, False, 1]
```

```
In [63]: # to excess last element also but it reverses the list
         list1[-1::-1]

Out[63]: [3, 1, False, True, 'Yellow', 'Green', 'Blue', 'Red']
```

```
In [64]: # range of elements
         l[-4:-1]

Out[64]: [(1, 2, 3), {1, 2}, {'key': 'value'}]
```

```
In [65]: # reverse of list
         list1[::-1]

Out[65]: [3, 1, False, True, 'Yellow', 'Green', 'Blue', 'Red']
```

3.0 Change list elements

```
In [66]: list1 = ["Red", "Blue", "Green","Yellow", True, False, 1,3]
```

3.1 Change item value at specific index

```
In [67]: # using list assignment
         list1[1] = 1
         list1

Out[67]: ['Red', 1, 'Green', 'Yellow', True, False, 1, 3]
```

3.2 Insert multiple values

```
In [68]: list1[2:5] = ["ramesh", "Suresh", "kamlesh"]
         list1

Out[68]: ['Red', 1, 'ramesh', 'Suresh', 'kamlesh', False, 1, 3]
```

```
In [69]: #if replacement items are not equal to items to be replaced then list length will increase or decrease
         print("length of list before replacement", len(list1))
         list1[1:3] = [22,34,45]
         print("length of list after replacement", len(list1))
         list1
```

```
length of list before replacement 8
length of list after replacement 9
```
Out[69]:
```
['Red', 22, 34, 45, 'Suresh', 'kamlesh', False, 1, 3]
```

In [70]:
```
print("length of list before replacement", len(list1))
list1[1:3] = ["sam"]
print("length of list after replacement", len(list1))
list1
```

```
length of list before replacement 9
length of list after replacement 8
```
Out[70]:
```
['Red', 'sam', 45, 'Suresh', 'kamlesh', False, 1, 3]
```

4.0 Add elements to list

In [89]:
```
l = [1,2,3,4,5]
```

4.1 append() Method

1. append() adds an item to the end of the list

In [90]:
```
l.append("iNeuron")
l
```

Out[90]:
```
[1, 2, 3, 4, 5, 'iNeuron']
```

4.2 insert() Method

1. To insert a new list item, without replacing any of the existing values, we can use the insert() method.
2. The insert() method inserts an item at the specified index.
3. This increases list length

In [91]:
```
l.insert(3,"Krish")
l
```

Out[91]:
```
[1, 2, 3, 'Krish', 4, 5, 'iNeuron']
```

4.3 extend() method

1. Any iterables can be added to list using extend() method
2. iterables: list, tuple, set, dictionary, etc

In [92]:
```
l.extend({"name","shubham", "job","developer"})
l
```

Out[92]:
```
[1, 2, 3, 'Krish', 4, 5, 'iNeuron', 'developer', 'job', 'shubham', 'name']
```

5.0 Remove elements from list

In [97]:
```
l = [1,2,3,4,5,6,7,8,9,10]
```

5.1 remove() method

1. It removes a specified element

In [98]:
```
l.remove(10)
l
```

Out[98]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5.2 pop()

1. It removes item at specified index
2. If no index specified pop() removes last element.

```
In [100…  l.pop(0)
          l

Out[100]: [2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [101…  l.pop()
          l

Out[101]: [2, 3, 4, 5, 6, 7, 8]
```

5.3 del keyword

1. It also removes element at specified index

```
In [102…  # at index 3 element value 5 will be removed
          del l[3]
          l

Out[102]: [2, 3, 4, 6, 7, 8]
```

5.4 clear() method

1. It is used to empty thr list.

```
In [104…  l.clear()
          l

Out[104]: []
```

6.0 Loop through lists

```
In [2]:  l = ["a",'b','c', 'd']
```

6.1 Using for loop

```
In [3]:  # for items without index
         for char in l:
             print(char)

a
b
c
d
```

```
In [4]:  # using index
         for i in range(len(l)):
             print(l[i])

a
b
c
d
```

6.2 Using while loop

```
In [5]:  i=0
         while i< len(l):
             print(l[i])
             i+=1

a
b
c
d
```

7.0 list comprehension

1. list comprehension are one line of code inside [ ] brackets

7.1 Syntax

1. [expression for item in iterable if condition == True]
2. The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list.
3. iterable can be list, tuple, set, etc
4. Condition is filter that only accepts items that evaluate to True.

7.2 Implementation with examples

In [14]:
```
colors = ["red", "blue", "yellow", "brown"]
```

In [15]:
```
# for condition and iterable
[x for x in colors if x not in ["red", "blue"]]
```

Out[15]:
```
['yellow', 'brown']
```

In [16]:
```
# generator function
[x for x in range(10)]
```

Out[16]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [17]:
```
# generator with condition
[x for x in range(8) if x<5]
```

Out[17]:
```
[0, 1, 2, 3, 4]
```

In [18]:
```
# expressions
[x.upper() for x in colors]
```

Out[18]:
```
['RED', 'BLUE', 'YELLOW', 'BROWN']
```

In [19]:
```
[x if x!='red' else 'No color for you' for x in colors]
```

Out[19]:
```
['No color for you', 'blue', 'yellow', 'brown']
```

In [23]:
```
# even no in list
l_int = [1,2,3,4,5,6,7,8,9,10]

[x if x%2==0 else "even" for x in l_int]
```

Out[23]:
```
['even', 2, 'even', 4, 'even', 6, 'even', 8, 'even', 10]
```

In [25]:
```
# better way to to even no example
[x for x in l_int if x%2==0]
```

Out[25]:
```
[2, 4, 6, 8, 10]
```

8.0 Sorting list element

In [42]:
```
l_alpha = ["Red", "Blue", "Green","Yellow"]
l_num = [1,4,2,5,3,6,8]
```

8.1 Using sort() method

In [44]:
```
l_alpha.sort()
l_alpha
```

Out[44]:
```
['Blue', 'Green', 'Red', 'Yellow']
```

In [46]:
```
l_alpha.sort(reverse=True)
l_alpha
```

Out[46]:
```
['Yellow', 'Red', 'Green', 'Blue']
```

In [47]:
```
l_num.sort()
l_num
```

Out[47]:
```
[1, 2, 3, 4, 5, 6, 8]
```

In [49]:
```
l_num.sort(reverse=True)
```

```
l_num
```

Out[49]:
```
[8, 6, 5, 4, 3, 2, 1]
```

8.2 Case insensitive sort

In [51]:
```
# case insensitive sort
l_alpha_upper_lower = ["red", "Blue", "green","Yellow"]
l_alpha_upper_lower.sort()
l_alpha_upper_lower
```

Out[51]:
```
['Blue', 'Yellow', 'green', 'red']
```

In [52]:
```
# now to make this sort case insensitive
l_alpha_upper_lower.sort(key=str.lower)
l_alpha_upper_lower
```

Out[52]:
```
['Blue', 'green', 'red', 'Yellow']
```

8.3 Custom sort functions

In [64]:
```
l = [('apple', 4), ('banana', 2), ('orange', 1), ('mango',3)]
```

In [65]:
```
# sorting based on numbers
sorted(l, key=lambda x:x[1])
```

Out[65]:
```
[('orange', 1), ('banana', 2), ('mango', 3), ('apple', 4)]
```

In [67]:
```
#sorting based on string
sorted(l, key=lambda x:x[0])
```

Out[67]:
```
[('apple', 4), ('banana', 2), ('mango', 3), ('orange', 1)]
```

9.0 Creating copy of list

Note: You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

For example see below code

In [71]:
```
l1 = [1,2,3,4,5]
l2 = l1
l2
```

Out[71]:
```
[1, 2, 3, 4, 5]
```

In [72]:
```
l1.insert(3, "shubham")
l2
```

Out[72]:
```
[1, 2, 3, 'shubham', 4, 5]
```

9.1 Using copy() method

In [75]:
```
l = [1,2,3,4,5]
l_copy = l.copy()

# inserting element in list l which will not affect copy list l_copy
l.insert(3,"shubham")
l_copy
```

Out[75]:
```
[1, 2, 3, 4, 5]
```

9.2 Using list() method

In [76]:
```
l1 = [1,2,3,4,5]
l_copy1 = list(l1)

# inserting element in list l1 which will not affect copy list l_copy1
l1.insert(3,"shubham")
l_copy1
```

```
Out[76]: [1, 2, 3, 4, 5]
```

10.0 Joining lists

10.1 Using `'+'` operator

```
In [77]: l1 = ["shubham"]
         l2 = ["verma"]
         l3 = l1 + l2
         l3
```

```
Out[77]: ['shubham', 'verma']
```

Note: creating duplicate items inside a list

```
In [79]: l3 = l2*4
         l3
```

```
Out[79]: ['verma', 'verma', 'verma', 'verma']
```

10.2 Using append() method

```
In [86]: l1 = [1,2,3]
         l2 = [2,3,4]
         for x in l2:
             l1.append(x)
         l1
```

```
Out[86]: [1, 2, 3, 2, 3, 4]
```

10.3 Using extend() method

```
In [90]: l1 = ["shubham"]
         l2 = ["verma"]
         l1.extend(l2)
         l1
```

```
Out[90]: ['shubham', 'verma']
```

11.0 List methods

1. append() Adds an element at the end of the list
2. clear() Removes all the elements from the list
3. copy() Returns a copy of the list
4. count() Returns the number of elements with the specified value
5. extend() Add the elements of a list (or any iterable), to the end of the current list
6. index() Returns the index of the first element with the specified value
7. insert() Adds an element at the specified position
8. pop() Removes the element at the specified position
9. remove() Removes the item with the specified value
10. reverse() Reverses the order of the list
11. sort() Sorts the list

All above methods except reverse and count are covered above

11.1 reverse() method

```
In [92]: l = [9,8,5,4,3,2,2,1]
         l.reverse()
         l
```

```
Out[92]: [1, 2, 2, 3, 4, 5, 8, 9]
```

11.2 count() method

In [93]:
```
l = [1,1,1,2,2,2,3,3,3,4,4,4,4,5,5,5,5]
l.count(1)
```

Out[93]:
```
3
```

In [96]:
```
# counting all elements
for i in set(l):
    print("Number: {} and its count {}".format(i,l.count(i)))
```

```
Number: 1 and its count 3
Number: 2 and its count 3
Number: 3 and its count 3
Number: 4 and its count 4
Number: 5 and its count 4
```