5. List

October 18, 2022

1 Introduction

- List is a heterogeneous collection which allows us to put many objects in a single object
- Lists are ordered and we have an index to access.
- Lists are created using square brackets.
- List items are ordered, changeable, and allow duplicate values.

[1]:
```
cities = ["bangalore", "mumbai", "delhi", "kolkata"]
```

[2]:
```
print(cities)
```

```
['bangalore', 'mumbai', 'delhi', 'kolkata']
```

[3]:
```
type(cities)
```

[3]: list

[4]:
```
even_nos = [2, 40, 16, 8, 10, 62, 74, 98, 32]
```

[5]:
```
type(even_nos)
```

[5]: list

[6]:
```
multi = ["hi", 78, True, 3.14, "sorry"]   # list can contain different types of␣
↪objects
```

[7]:
```
type(multi)
```

[7]: list

2 Length of List

- Use `len` function

[8]:
```
len(cities)
```

[8]: 4

[9]:
```
len(even_nos)
```

[9]:
```
9
```

3 Empty List

[10]:
```
el = list()
```

[11]:
```
len(el)
```

[11]:
```
0
```

[12]:
```
type(el)
```

[12]:
```
list
```

[13]:
```
 el2 = []
```

4 Accessing elements of the List

- Elements of the List can be accessed using **indexing**
- We can use the index operator `[]` (square brackets) to access an item in a list.
- In Python, indices start at `0`(zero). So, a list having 5 elements will have an index from 0 to 4.
- Trying to access indexes other than these will raise an `IndexError`.
- The index must be an integer. We can't use float or other types, this will result in `TypeError`.
- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

[14]:
```
cities
```

[14]:
```
['bangalore', 'mumbai', 'delhi', 'kolkata']
```

[15]:
```
cities[0]
```

[15]:
```
'bangalore'
```

[16]:
```
cities[1]
```

[16]:
```
'mumbai'
```

[17]:
```
cities[2]
```

[17]:
```
'delhi'
```

[18]:
```
cities[3]
```

[18]:
```
'kolkata'
```

[19]:
```
cities[-1]
```

[19]:
```
'kolkata'
```

[20]:
```
cities[-2]
```

[20]:
```
'delhi'
```

[21]:
```
cities[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [21], line 1
----> 1 cities[4]

IndexError: list index out of range
```

[22]:
```
cities[-5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [22], line 1
----> 1 cities[-5]

IndexError: list index out of range
```

5 List Slicing

- We can access a range of items in a list by using the slicing operator : (colon).

[23]:
```
even_nos
```

[23]:
```
[2, 40, 16, 8, 10, 62, 74, 98, 32]
```

[24]:
```
len(even_nos)
```

[24]:
```
9
```

[26]:
```
even2 = even_nos[2:8]
```

[27]:
```
even2
```

[27]:
```
[16, 8, 10, 62, 74, 98]
```

[28]:
```
even_nos[:6]
```

[28]:
```
[2, 40, 16, 8, 10, 62]
```

[29]:
```
even_nos[3:]
```

[29]:
```
[8, 10, 62, 74, 98, 32]
```

6 Modifying a List

- Lists are mutable, meaning their elements can be changed
- We can use the assignment operator = to change an item or a range of items.

[30]:
```
cities
```

[30]:
```
['bangalore', 'mumbai', 'delhi', 'kolkata']
```

[31]:
```
cities[2] = 'pune'
```

[32]:
```
cities
```

[32]:
```
['bangalore', 'mumbai', 'pune', 'kolkata']
```

6.1 Modify using index range

[34]:
```
cities
```

[34]:
```
['bangalore', 'mumbai', 'pune', 'kolkata']
```

[35]:
```
cities[1:3]
```

[35]:
```
['mumbai', 'pune']
```

[36]:
```
cities[1:3] = ["chennai", "jaipur"]
```

[37]:
```
cities
```

[37]:
```
['bangalore', 'chennai', 'jaipur', 'kolkata']
```

7 Properties of Lists

1. Heterogeneous (any data type!)
2. Ordered (numbered from 0 to n-1)
3. Have random access to any element (Using index)
4. Number of elements can change very easily.
5. Lists are mutable

8 Methods of List

1. append(elem)
2. extend(L)
3. insert(i, elem)
4. remove(elem)
5. pop([i])
6. clear()
7. index(elem)
8. count(elem)
9. sort()
10. reverse()

8.1 append(elem)

- Add item elem at the end of the list

[38]:
```
cities
```

[38]:
```
['bangalore', 'chennai', 'jaipur', 'kolkata']
```

[39]:
```
cities.append("pune")
```

[40]:
```
cities
```

[40]:
```
['bangalore', 'chennai', 'jaipur', 'kolkata', 'pune']
```

8.2 extend(L)

- Add all items in given list L to the end

[41]:
```
cities2 = ["mysore", "nagpur", "bhopal"]
```

[42]:
```
cities
```

[42]:
```
['bangalore', 'chennai', 'jaipur', 'kolkata', 'pune']
```

[43]:
```
cities.extend(cities2)
```

[44]:
```
cities
```

[44]:
```
['bangalore',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
```

```
 'bhopal']
```

8.3 insert(i, elem)

- Insert item elem at position `i`

[50]:
```
cities
```

[50]:
```
['bangalore',
 'mumbai',
 'mumbai',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat',
 'bhopal']
```

[48]:
```
cities.insert(1, "mumbai")
```

[49]:
```
cities.insert(-1, "Surat")
```

[51]:
```
cities
```

[51]:
```
['bangalore',
 'mumbai',
 'mumbai',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat',
 'bhopal']
```

8.4 remove(elem)

- Remove first item that is equal to elem, from the list

[52]:
```
cities
```

[52]:
```
['bangalore',
 'mumbai',
 'mumbai',
```

```
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat',
 'bhopal']
```

[53]:
```
cities.remove('mumbai')
```

[54]:
```
cities
```

[54]:
```
['bangalore',
 'mumbai',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat',
 'bhopal']
```

8.5 pop([i])

- Remove and return item at position i (last item if i is not provided)

[55]:
```
cities
```

[55]:
```
['bangalore',
 'mumbai',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat',
 'bhopal']
```

[56]:
```
cities.pop()
```

[56]:
```
'bhopal'
```

[57]:
```
cities
```

[57]:
```
['bangalore',
 'mumbai',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat']
```

[58]:
```
cities.pop(3)
```

[58]:
```
'jaipur'
```

[59]:
```
cities
```

[59]:
```
['bangalore',
 'mumbai',
 'chennai',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'Surat']
```

[60]:
```
c = cities.pop()
```

[61]:
```
c
```

[61]:
```
'Surat'
```

8.6 index(elem)

- Return index of first item that is equal to elem

[66]:
```
cities
```

[66]:
```
['bangalore',
 'mumbai',
 'chennai',
 'kolkata',
 'pune',
 'mysore',
 'nagpur',
 'mumbai']
```

[67]:
```
cities.index('mumbai')
```

[67]: 1

[64]:
```
cities.index('chennai')
```

[64]: 2

[65]:
```
cities.append("mumbai")
```

8.7 count(elem)

- Return the number of items that is equal to elem

[68]:
```
cities.count('mumbai')
```

[68]: 2

8.8 sort()

- Sort items in a list in ascending order
- To sort in descending order use `reverse = True`

[71]:
```
cities
```

[71]:
```
['bangalore',
 'chennai',
 'kolkata',
 'mumbai',
 'mumbai',
 'mysore',
 'nagpur',
 'pune']
```

[70]:
```
cities.sort()
```

[73]:
```
cities
```

[73]:
```
['pune',
 'nagpur',
 'mysore',
 'mumbai',
 'mumbai',
 'kolkata',
 'chennai',
 'bangalore']
```

[80]:
```
cities.sort(reverse = True)
```

[74]:
```
cities
```

[74]:
```
['pune',
 'nagpur',
 'mysore',
 'mumbai',
 'mumbai',
 'kolkata',
 'chennai',
 'bangalore']
```

8.9 reverse()

- Reverse the order of items in a list

[118]:
```
cities = ['bangalore',
 'mumbai',
 'Surat',
 'chennai',
 'jaipur',
 'kolkata',
 'pune',
 'nagpur',
 'Surat', 'pune']
```

[121]:
```
cities
```

[121]:
```
['bangalore',
 'mumbai',
 'Surat',
 'chennai',
 'jaipur',
 'kolkata',
 'nagpur',
 'Surat',
 'pune']
```

[76]:
```
cities.reverse()
```

[77]:
```
cities
```

[77]:
```
['Surat',
 'nagpur',
 'pune',
 'kolkata',
 'jaipur',
 'chennai',
```

```
 'Surat',
 'mumbai',
 'bangalore']
```

8.10 clear()

- Remove all items and empty the list

[78]:
```
cities.clear()
```

[79]:
```
cities
```

[79]:
```
[]
```

9 Deleting a List

- Use del

[81]:
```
even_nos
```

[81]:
```
[2, 40, 16, 8, 10, 62, 74, 98, 32]
```

[82]:
```
del even_nos
```

[83]:
```
even_nos
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [83], line 1
----> 1 even_nos

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

10 Operators on List

1. Membership
2. Arithmetic

10.1 Membership

- in
- not in

[84]:
```
multi
```

[84]:
```
['hi', 78, True, 3.14, 'sorry']
```

[85]:
```
78 in multi
```

[85]:
```
True
```

[86]:
```
40 not in multi
```

[86]:
```
True
```

[87]:
```
'hi' not in multi
```

[87]:
```
False
```

[88]:
```
True in multi
```

[88]:
```
True
```

10.2 Arithmetic

- +
- *

[89]:
```
p = [1, 2, 3]
q = [4, 5, 6]
```

[90]:
```
p + q
```

[90]:
```
[1, 2, 3, 4, 5, 6]
```

[91]:
```
p * 3
```

[91]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

11 Nested Lists

- List as a List element or we can say list inside a list
- Nested lists are accessed using nested indexing.

[92]:
```
nestl = ["Monday", "Today", [1, 2, 3]]
```

[93]:
```
nestl[0]
```

[93]:
```
'Monday'
```

[94]:
```
nestl[2]
```

[94]:
```
[1, 2, 3]
```

[95]:
```
nestl[2][0]
```

[95]:
```
1
```

A matrix can be represented using a nested list

[96]:
```
mx = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[97]:
```
mx
```

[97]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[98]:
```
mx[0][2]
```

[98]:
```
3
```

12 `split` and `join` methods of String

12.1 `split()`

- The `split()` method splits a string into a **list**.
- You can specify the separator with `sep`, default separator is any whitespace.
- Maximum number of splits to do can be specified with `maxsplit`.

[99]:
```
sentence = "When the winter comes, the lone wolf dies but the pack survives."
```

[101]:
```
words = sentence.split()
```

[102]:
```
words
```

[102]:
```
['When',
 'the',
 'winter',
 'comes,',
 'the',
 'lone',
 'wolf',
 'dies',
 'but',
 'the',
 'pack',
 'survives.']
```

[103]:
```
sentence
```

[103]:
```
'When the winter comes, the lone wolf dies but the pack survives.'
```

[104]:
```
sentence.split(",")
```

[104]:
```
['When the winter comes', ' the lone wolf dies but the pack survives.']
```

[105]:
```
sentence.split(maxsplit=2) #maxsplit: Maximum number of splits to do.
```

[105]:
```
['When', 'the', 'winter comes, the lone wolf dies but the pack survives.']
```

12.2 join()

- The `join()` method takes all items in an iterable and joins them into one string.
- A string must be specified as the separator.
- Syntax: `string.join(iterable)`

[106]:
```
sentence_list = ['When',
 'the',
 'winter',
 'comes,',
 'the',
 'lone',
 'wolf',
 'dies',
 'but',
 'the',
 'pack',
 'survives.']
```

[111]:
```
" ".join(sentence_list)
```

[111]:
```
'Whenhithehiwinterhicomes,hithehilonehiwolfhidieshibuthithehipackhisurvives.'
```

[108]:
```
seperator = " "
```

[109]:
```
seperator.join(sentence_list)
```

[109]:
```
'When the winter comes, the lone wolf dies but the pack survives.'
```

13 For Loop on List

[112]:
```
sentence_list
```

[112]:
```
['When',
 'the',
 'winter',
 'comes,',
 'the',
```

```
 'lone',
 'wolf',
 'dies',
 'but',
 'the',
 'pack',
 'survives.']
```

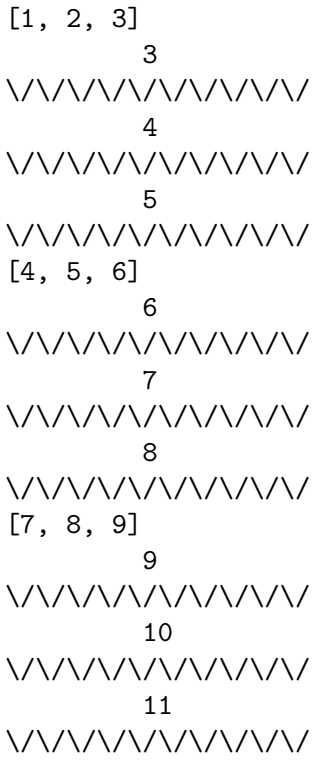
[113]:
```
for i in sentence_list:
    print(i)
    print('.-'*4)
```

```
When
.-.-.-.-
the
.-.-.-.-
winter
.-.-.-.-
comes,
.-.-.-.-
the
.-.-.-.-
lone
.-.-.-.-
wolf
.-.-.-.-
dies
.-.-.-.-
but
.-.-.-.-
the
.-.-.-.-
pack
.-.-.-.-
survives.
.-.-.-.-
```

13.1 Nested For Loop

[114]:
```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

[117]:
```
for i in matrix:
    print(i)
    for j in i:
        print('\t',j+2)
        print('\/'*10)
```


[1, 2, 3]
3
4
5
[4, 5, 6]
6
7
8
[7, 8, 9]
9
10
11