Lists & Tuples - Basic Properties

August 2, 2021

1 List

Creating a list...

[7]:
```
myList = [4, 2, 7, 5, 9]
myList
```

[7]:
```
[4, 2, 7, 5, 9]
```

1.1 Referring to elements of a list

Note that a list's indices begin from 0.

[5]:
```
myList = [4, 2, 7, 5, 9]
i = int(input("Enter index: "))
myList[i]
```

```
Enter index: 4
```

[5]:
```
9
```

1.2 List properties

Also note that a list can be of any data type, and it need not be homogenous.

[33]:
```
myList = ["Meow", 1, 5.4, 'c']
```

[34]:
```
myList
```

[34]:
```
['Meow', 1, 5.4, 'c']
```

Also note that a list can contain a collection data type object (ex. another list or tuple) as a single element.

[38]:
```
myList = [[1, 2, 3], "Hello", "World"]
myList
```

[38]:
```
[[1, 2, 3], 'Hello', 'World']
```

[39]:
```
myList[0]
```

[39]:
```
[1, 2, 3]
```

2 Tuple

Creating a tuple is similar to creating a list, but we use parentheses instead of square brackets. Also, append method does not apply for a tuple. Hence, while lists are mutable, tuples are immutable. Note that tuple values cannot be altered and its memory spaces cannot be allocated or freed, while for a list both are possible.

[ ]:
```
myTuple = (1, 2, 4)
myTuple
```

2.1 Referring to tuple elements

[8]:
```
myTuple = (0, 1, 4, 9, 16, 25)
i = int(input("Enter index: "))
myTuple[i]
```

```
Enter index: 5
```

[8]:
```
25
```

2.2 Tuple properties

Like a list, the data types of the elements of a tuple need not be homogenous.

[9]:
```
myTuple = ("Pranav", 1, 3.2)
myTuple
```

[9]:
```
('Pranav', 1, 3.2)
```

Like a list, a tuple can contain a collection data type object (ex. another list or tuple) as a single element.

3 Iterating over a list or tuple

Other than the definition of the object, everything is the same.

[44]:
```
myList = [1, 4, 2, 5, 4, "Boohoo", "Hsss"]
for i in myList:
    print(i)
```

```
1
4
2
```

```
5
4
Boohoo
Hsss
```

3.1 Iterating with indices

Other than the definition of the object, everything is the same.

[47]:
```
for index, item in enumerate(myList):
    print(index, item)
```

```
0 1
1 4
2 2
3 5
4 4
5 Boohoo
6 Hsss
```

4 Packing and unpacking

This involves assigning multiple list/tuple values into multiple other variables at once. It works in the same manner for both list or tuple. Packing refers to grouping multiple variables together, as if in a list or tuple. Unpacking refers to assigning each value of the list or tuple to the corresponding variable in the group of variables. Packing (variables a, b, c and d)… [a, b, c, d] Unpacking (list A)… [a, b, c, d] = A

[14]:
```
myList = ["Apple", "Mango", "Bottle Gourd", "Blueberry"]
[a, b, c, d] = myList
print(a, b, c, d)
```

```
Apple Mango Bottle Gourd Blueberry
```

[18]:
```
myTuple = ("Apple", "Mango", "Bottle Gourd", "Blueberry")
[a, b, c, d] = myTuple
print(a, b, c, d)
```

```
Apple Mango Bottle Gourd Blueberry
```

Packing can be done either in parantheses or in square brackets.

[19]:
```
[a, b, c, d] = myList
(a, b, c, d) = myList

[a, b, c, d] = myTuple
(a, b, c, d) = myTuple
```

4.1 Error cases

When you try to assign too many variables for a list.

[15]:
```
[a, b, c] = myList
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-45be8488af3c> in <module>
----> 1 [a, b, c] = myList

ValueError: too many values to unpack (expected 3)
```

[16]:
```
[a, b, c, d, e] = myList
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-a6b9af832bb0> in <module>
----> 1 [a, b, c, d, e] = myList

ValueError: not enough values to unpack (expected 5, got 4)
```