Python
Lists, Tuples, Sets, Dictionaries

Lab-2
5-02-2024
By Ms. Soofi Shafiya
NET-JRF, JMI

Python Collections (Arrays)

There are four collection data types in the Python programming language:

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.

Dictionary is a collection which is ordered** and changeable. No duplicate members.

Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

List items are **ordered**, **changeable**, and **allow duplicate values**.

Ordered

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.

Changeable

- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

- Since lists are indexed, lists can have items with the same value:

```
thislist =
["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

List Length

```
thislist = ["apple", "banana", "cherry"]
print(???(thislist))
```

List Items - Data Types

List items can be of any data type

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

print(????(list1))
```

The list() Constructor

- It is also possible to use the list() constructor when creating a new list.

- `thislist = list(("apple", "banana", "cherry"))`

```
# note the double round-brackets

print(thislist)
```

Python - Access List Items

Access items

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Negative Indexing

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexing

```
Lst=["apple", "banana", "cherry", "orange", "kiwi", "me
lon", "mango"]
print(Lst[2:5])
print(thislist[:4])
print(thislist[2:])
print(thislist[-4:-1])
```

Check if Item Exists

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Change List Items

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
Change the Range of values
thislist[1:3] = ["blackcurrant", "watermelon"]
```

Change the second value by replacing it with *two* new values:

```
thislist[1:2] = ["blackcurrant", "watermelon"]
```

If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

```
O/P: [‘apple’, ‘watermelon’]
```

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Add List Items

```
thislist = ["apple", "banana", "cherry"]

thislist.append("orange")  #to add an item to the end of the list
thislist.insert(1, "orange") #To insert a list item at a specified index
```

#To append elements from *another list* to the current list, use the extend() method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical) #add items of tropical to thislist
```

Add Any Iterable

```
thislist = ["apple", "banana", "cherry"] #List
thistuple = ("kiwi", "orange") #Tuple
thislist.extend(thistuple)
```

Remove Specified Item #If there are more than one item with the specified value, the remove() method removes the first occurance:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
```

Remove Specified Index

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1) # the pop() method removes the specified index.
print(thislist)

#If you do not specify the index, the pop() method
removes the last item.

#the del keyword also removes the specified index:
del thislist[0]
print(thislist)

del thislist #delete the complete list
```

Clear the List

- The clear() method empties the list.
- The list still remains, but it has no content.

- `thislist.clear()`

Python - Loop Lists

Loop Through a List

- You can loop through the list items by using for loop

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Loop Through the Index Numbers

#Use the range() and len() functions to create a suitable iterable

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Exercise:

WAP to print the elements of list using while Loop.

Program

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

List Comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

- Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Using for:

```
fruits =
["apple", "banana", "cherry", "kiwi", "mango"]#list

newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

Using list comprehension:

```
newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

Example:

newlist = [x for x in fruits if x != ”apple”]

- The *condition* is optional and can be omitted:

```
newlist = [x for x in fruits]
```

Iterable

- The *iterable* can be any iterable object, like a list, tuple, set etc.
- We can use range() function to create iterable:

```
newlist = [x for x in range(10)]
```

Output:????

Exercise:

Accept only numbers lower than 5.

Expression

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:

Exercise:

1) Set the values in the new list to upper case
2) Set the values in the new list to “hello”
3) Return "orange" instead of "banana"

Solutions

```
1. newlist = [x.upper() for x in fruits]
2. newlist = ['hello' for x in fruits]
3. newlist =
  [x if x != "banana" else "orange" for x in fruits]
```

Sort Lists

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
```

O/P?????

Note:

To sort descending, use the keyword argument reverse = True

Customize Sort Function

- You can also customize your own function by using the keyword argument *key = function*
- The function will return a number that will be used to sort the list (the lowest number first):

Example

- Sort the list based on how close the number is to 50:

```
def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)
```

- By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters

- So if you want a case-insensitive sort function, use str.lower as a key function:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)


thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse() #Reverse the order of the list items
```

Copy a List

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.

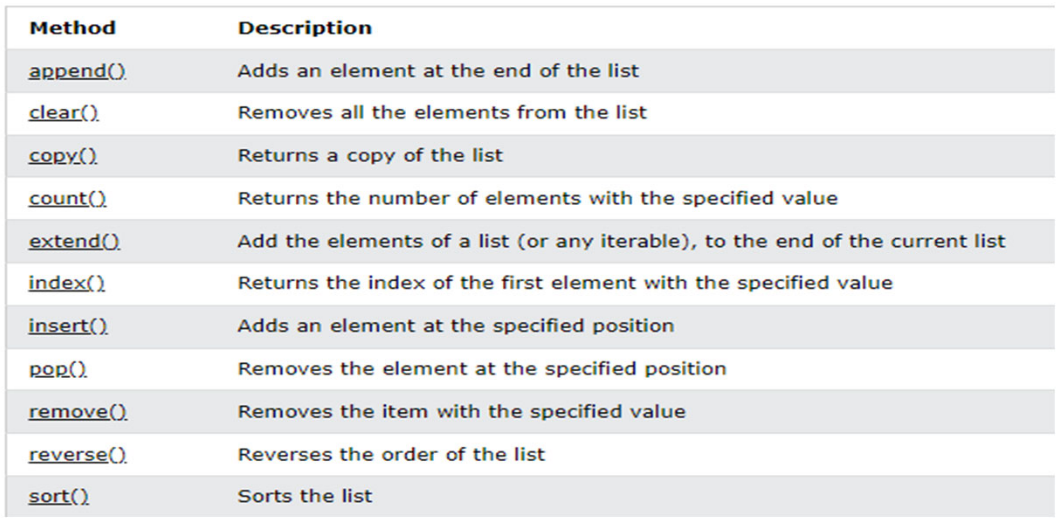
```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy() #using copy() method
print(mylist)


thislist = ["apple", "banana", "cherry"]
mylist = list(thislist) #using constructor list
print(mylist)
```

Join Lists

```
1. list3 = list1 + list2
2. for x in list2:
     list1.append(x)
3. list1.extend(list2)
```

List Methods



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

List Exercises

fruits=[‘banana’,’cherry’,’orange’,’apple’]

1. Print the second item in the fruits list
2. Change the value from "apple" to "kiwi", in the fruits list.
3. Use “append” method to add ‘orange’ in the list
4. Use the “insert” method to add ‘lemon’ as the second item of the list

5. Use “remove” method to remove banana from list
6. Use negative indexing to print the last item in the list.
7. Use a range of indexes to print the third, fourth, and fifth item in the list.
8. Use the correct syntax to print the number of items in the list.