PYTHON INTRODUCTION – 28NOV2020

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Example

Create a List:

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

List Items

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

List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

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.

Note: There are some list methods that will change the order, but in general: the order of the items will not change.

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:

List Length

To determine how many items a list has, use the `len()` function:

Example

Print the number of items in the list:

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

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

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

Try it Yourself »

A list can contain different data types:

Example

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

type()

From Python's perspective, lists are defined as objects with the data type 'list':

```
<class 'list'>
```

Example

What is the data type of a list?

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

The list() Constructor

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

Example

Using the `list()` constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

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 and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered and changeable. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Access Items

List items are indexed and you can access them by referring to the index number:

Example

Print the second item of the list:

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

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end

`-1` refers to the last item, `-2` refers to the second last item etc.

Example

Print the last item of the list:

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

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT included, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" and to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "orange" (-4) to, but NOT included. "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

Example

Check if "apple" is present in the list:

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

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

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

To insert more than one item, create a list with the new values, and specify the index number where you want the new values to be inserted:

Example

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

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

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

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

Insert Items

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

The `insert()` method inserts an item at the specified index:

Example

Insert "watermelon" as the third item:

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

Note: As a result of the example above, the list will now contain 4 items.

Append Items

To add an item to the end of the list, use the `append()` method:

Example

Using the `append()` method to append an item:

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

```
['apple', 'banana', 'cherry', 'orange']
```

Insert Items

To insert a list item at a specified index, use the `insert()` method.

The `insert()` method inserts an item at the specified index:

Example

Insert an item as the second position:

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

```
['apple', 'orange', 'banana', 'cherry']
```

Extend List

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

Example

Add the elements of `tropical` to `thislist`:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

print(thislist)
```

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

The elements will be added to the *end* of the list.

Add Any Iterable

The `extend()` method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")

thislist.extend(thistuple)

print(thislist)
```

```
['apple', 'banana', 'cherry', 'kiwi', 'orange']
```

Remove Specified Item

The `remove()` method removes the specified item.

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

```
['apple', 'cherry']
```

Remove Specified Index

The `pop()` method removes the specified index.

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

```
['apple', 'cherry']
```

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

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

```
['apple', 'banana']
```

The `del` keyword also removes the specified index:

Example

Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

The `del` keyword can also delete the list completely.

Example

Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist) #this will cause an error because you have succsesfully deleted "thislist".
```

```
Traceback (most recent call last):
  File "demo_list_del2.py", line 3, in <module>
    print(thislist) #this will cause an error because you have succsesfully deleted "thislist".
NameError: name 'thislist' is not defined
```

Clear the List

The `clear()` method empties the list.

The list still remains, but it has no content.

Example

Clear the list content:

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

```
[]
```

Loop Through a List

You can loop through the list items by using a `for` loop:

Example

Print all items in the list, one by one:

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

```
apple
banana
cherry
```

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

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

Example

Print all items by referring to their index number:

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

```
apple
banana
cherry
```

The iterable created in the example above is `[0, 1, 2]`.

Using a While Loop

You can loop through the list items by using a `while` loop.

Use the `len()` function to determine the length of the list, then start at 0 and loop your way through the list items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a `while` loop to go through all the index numbers

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

```
apple
banana
cherry
```

Looping Using List Comprehensive

List Comprehensive offers the shortest syntax for looping through lists:

Example

Print all items, using a `while` loop to go through all the index numbers

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

```
apple
banana
cherry
```

https://www.w3schools.com/python/python_lists_loop.asp

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.

Without list comprehension you will have to write a `for` statement with a conditional test inside:

Example

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

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

print(newlist)
```

```
['apple', 'banana', 'mango']
```

With list comprehension you can do all that with only one line of code:

Example

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

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

print(newlist)
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]

print(newlist)
```

```
['apple', 'banana', 'mango']
```

The Syntax

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

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that accepts only the items that valuates to `True`.

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

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

print(newlist)
```

```
['banana', 'cherry', 'kiwi', 'mango']
```

Difference between List and Array in Python

Last Updated: 17-07-2020

List: A list in Python is a collection of items which can contain elements of multiple data types, which may be either numeric, character logical values, etc. It is an ordered collection supporting negative indexing. A list can be created using [] containing data values.

Contents of lists can be easily merged and copied using python's inbuilt functions.

```
# creating a list containing elements
# belonging to different data types
sample_list = [1,"Yash",['a','e']]
print(sample_list)
```

Output :

```
[1, 'Yash', ['a', 'e']]
```

The first element is an integer, the second a string and the third is an list of characters.

Array: An array is a vector containing homogeneous elements i.e. belonging to the same data type. Elements are allocated with contiguous memory locations allowing easy modification, that is, addition, deletion, accessing of elements. In Python, we have to use the `array` module to declare arrays. If the elements of an array belong to different data types, an exception "Incompatible data types" is thrown.

```
# creating an array containing same
# data type elements
import array

sample_array = array.array('i', [1, 2, 3])

# accessing elements of array
for i in sample_array:
     print(i)
```

LIST	ARRAY
Can consist of elements belonging to different data types	Only consists of elements belonging to the same data type
No need to explicitly import a module for declaration	Need to explicitly import a module for declaration
Cannot directly handle arithmetic operations	Can directly handle arithmetic operations
Can be nested to contain different type of elements	Must contain either all nested elements of same size
Preferred for shorter sequence of data items	Preferred for longer sequence of data items
Greater flexibility allows easy modification (addition, deletion) of data	Less flexibility since addition, deletion has to be done element wise
The entire list can be printed without any explicit looping	A loop has to be formed to print or access the components of array
Consume larger memory for easy addition of elements	Comparatively more compact in memory size

Arrays in Python

(1)

```
from array import *
array1 = array('i', [10, 20, 30, 40, 50])
array1[0]
print(len(array1))
```

(2)

```
# Python program to demonstrate  Creation of Arrays

# importing "array" for array creations
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ", end = "      ")
for i in range (0, len(a)):
    print (a[i], end ="\t")

print("\n")

print("The length of the integer array before insertion is:   ", len(a))

#insert function for array element insertion
a.insert(0,0)
print("\n")

print("The length of the integer array after insertion is:   ", len(a))

print("\n")

print ("The array after insertion of an element is : ", end = "      ")
for i in range (0, len(a)):
    print (a[i], end ="\t")

print("\n")

# adding an element using append()
a.append(4)
```

```
print ("The array after appending an element is : ", end = "     ")
for i in range (0, len(a)):
    print (a[i], end ="\t")

print("\n")

#print the index of an element in the array
ind1 = a.index(3)

print("index of element 3 in array = ", ind1)

print(“The array before sorting is: “)
for i in range (0, len(a)):
    print (a[i], end ="\t")

print(“The array after sorting is: “)
for i in range (0, len(a)):
    print (a[i], end ="\t")
```

```
import array as arr
import numpy as np

# creating an array with integer type
a = arr.array('i', [3, 1, 2])

# printing original array
print ("The new created array is : ", end = "     ")
for i in range (0, len(a)):
    print (a[i], end ="\t")

print("\n")

arr1 = np.array(a)

print(np.sort(arr1))

#a.sort()
```

```
The new created array is :      3       1       2

[1 2 3]
```

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

```
[0 1 2 3]
```