Python Tutorial
Data Types

S1 MCA

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

- 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:

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.

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:

Lists allow duplicate values:

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

List Length

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

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:
- A list can contain different data types:

String, int and boolean data types:

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

A list with strings, integers and boolean values:

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

Try it Yourself »

- **type()**

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

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.

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

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

Python - Access List Items

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

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

→ will print second item of the list

- **Negative Indexing**

Negative indexing means start from the end

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

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

→Print the last Item of the list

- **Range of Indexes**

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

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

→Return the third, fourth, and fifth item:

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

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

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

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

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

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

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

- **Range of Negative Indexes**

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

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

→returns the items from "orange" (-4) to, but NOT including "mango" (-1):

- Check if Item Exists

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

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

→Check if "apple" is present in the list:

Python - Change List Items

- **Change Item Value**

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

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

→ Change the second item:

- **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

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

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

- If you insert *more* 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[1:2] = ["blackcurrant", "watermelon"]
     print(thislist)
```

→output:['apple', 'blackcurrant', 'watermelon', 'cherry']

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

- 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)
```

Change the second and third value by replacing it with *one* value:

['apple', 'watermelon’]

Python - Add List Items

- **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

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

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

→Output:['apple', 'banana', 'watermelon', 'cherry’]

- **Append Items**

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

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

- **Extend List**

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)
       print(thislist)
```

→Output:['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

Python - Remove List Items

- **Remove Specified Item**

The remove() method removes the specified item.

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

→Output:['apple', 'cherry']

→If there are more than one item with the specified value, the remove() method removes the first occurrence:

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

→output:['apple', 'cherry', 'banana', 'kiwi’]

- **Remove Specified Index**

The pop() method removes the specified index.

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

→Output:['apple', 'cherry']

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

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

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

→Output:['apple', 'banana’]

- The **del** keyword also removes the specified index:

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

→Output : ['banana', 'cherry’]

- The del keyword can also delete the list completely

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

- **Clear the List**

The clear() method empties the list.

The list still remains, but it has no content.

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

Python - Loop Lists

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

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

- **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.

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

- **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 referring to their indexes.

Remember to increase the index by 1 after each iteration**.**

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

Python - List Comprehension

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

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

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

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

print(newlist)
```

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

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

item – a variable that takes each value from iterable.

expression – what you want to put into the new list (often item, but can also be something transformed like item.upper()).

if condition == True – a filter: only items where the condition is True are included.

You can shorten if condition == True to just if condition.

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

       newlist = [x.upper() for x in fruits if "a" in x]

       print(newlist)
```

→Output: ['APPLE', 'BANANA', 'MANGO’]

You can use the range() function to create an iterable:

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

Sort List

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

```
>>>  thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
     thislist.sort()
     print(thislist)
>>>  thislist = [100, 50, 65, 82, 23]
     thislist.sort()
     print(thislist)
```

- **Sort Descending**

To sort descending, use the keyword argument reverse = True

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


>>>    thislist = [100, 50, 65, 82, 23]
       thislist.sort(reverse = True)
       print(thislist)
```

- Customize Sort Function

You can also customize your own function by using the keyword argument key = function.

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

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

- Case Insensitive Sort

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

```
>>>    thislist = ["banana", "Orange", "Kiwi", "cherry"]
       thislist.sort()
       print(thislist)
```

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

- **Reverse Order**

The reverse() method reverses the current sorting order of the elements.

```
>>>    thislist = ["banana", "Orange", "Kiwi", "cherry"]
       thislist.reverse()
       print(thislist)
```

→ Output :['cherry', 'Kiwi', 'Orange', 'banana']

- **String and number together**

```
>>>    thislist = ["orange", "mango", "kiwi", "pineapple", 1]
       thislist = [str(item) for item in thislist]
       thislist.sort()
       print(thislist)
```

→ Output :['1', 'kiwi', 'mango', 'orange', 'pineapple']

```
>>>    thislist = ["orange", "mango", "kiwi", "pineapple", 1]
      strings = sorted([x for x in thislist if isinstance(x, str)])
      numbers = sorted([x for x in thislist if isinstance(x, int)])
      sorted_list = numbers + strings
      print(sorted_list)
```

→ Output : [1, 'kiwi', 'mango', 'orange', 'pineapple']

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.

- You can use the built-in List method copy() to copy a list.
- >>> thislist = ["apple", "banana", "cherry"]
  mylist = thislist.copy()
  print(mylist)

- **Use the list() method**

Another way to make a copy is to use the built-in method list().

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

- **Use the slice Operator**

You can also make a copy of a list by using the : (slice) operator.

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

Join Two Lists

- There are several ways to join, or concatenate, two or more lists in Python.
- One of the easiest ways are by using the + operator.

```
>>>    list1 = ["a", "b", "c"]
       list2 = [1, 2, 3]

       list3 = list1 + list2
       print(list3)
```

- Another way to join two lists is by appending all the items from list2 into list1, one by one:

```
>>>    list1 = ["a", "b" , "c"]
       list2 = [1, 2, 3]

       for x in list2:
              list1.append(x)

       print(list1)
```

- you can use the extend() method, where the purpose is to add elements from one list to another list

```
>>>    list1.extend(list2)
       print(list1)
```

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