Training Report Day-3

8 June 2024

❖ LISTS IN PYTHON:-

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:

```
FRUITS = ["apple", "banana", "cherry", 24, 78.90, True, ["ram", "sham"]]
print(FRUITS)
```

```
Students = []
Students
```

Some of the more relevant characteristics of list objects include being:

Ordered: They contain elements or items that are sequentially arranged according to their specific insertion order.

Zero-based: They allow you to access their elements by indices that start from zero.

Mutable: They support in-place mutations or changes to their contained elements.

Heterogeneous: They can store objects of different types.

Grow able and dynamic: They can grow or shrink dynamically, which means that they support the addition, insertion, and removal of elements.

Nestable: They can contain other lists, so you can have lists of lists.

Iterable: They support iteration, so you can traverse them using a loop or comprehension while you perform operations on each of their elements.

Sliceable: They support slicing operations, meaning that you can extract a series of elements from them.

Combinable: They support concatenation operations, so you can combine two or more lists using the concatenation operators.

Copy able: They allow you to make copies of their content using various techniques.

Lists are sequences of objects. They're commonly called containers or collections because a single list can contain or collect an arbitrary number of other objects.

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, the third item has index [2] etc.
keyboard_arrow_down

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

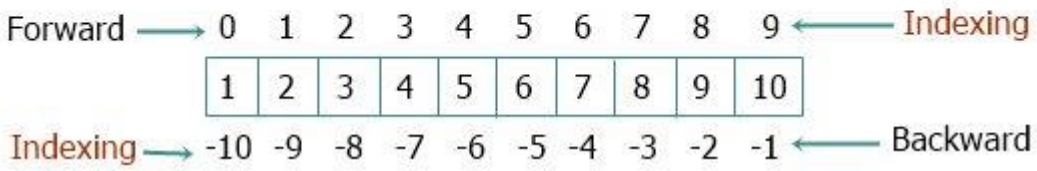
Since lists are indexed, lists can have items with the same value:

```
#Example
#Lists allow duplicate values:
my_list = ["apple", "banana", "cherry", "apple", "cherry"]
print(my_list)
```

```
#Example
#Using the list() constructor to make a List:
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

- **List indexing:-**

```
mylist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```




```
#LIST METHODS
#APPEND: Used to new elements to the end of an existing list.
#syntax = list_name.append("item")
UNI = ["CTU", "KTU", "DTU"]
UNI.append("CU")
print(UNI)
```

```
#sorting a list
# by using sort() method
FUR.sort()
print(FUR)
#by using sorted() function
FUR = ["TABLE", "CHAIR", "STOOL", "BED"]
print(sorted(FUR))
```

In general, you should use lists when you need to:

Keep your data ordered: Lists maintain the order of insertion of their items.

Store a sequence of values: Lists are a great choice when you need to store a sequence of related values.

Mutate your data: Lists are mutable data types that support multiple mutations.

Access random values by index: Lists allow quick and easy access to elements based on their index.

In contrast, avoid using lists when you need to:

Store immutable data: In this case, you should use a tuple. They're immutable and more memory efficient.

Represent database records: In this case, consider using a tuple or a data class.

Store unique and unordered values: In this scenario, consider using a set or dictionary. Sets don't allow duplicated values, and dictionaries can't hold duplicated keys.

➢ LIST SLICING:

List slicing in Python allows you to extract a portion of a list by specifying a range of indices. The general syntax for list slicing is:

Start: The index where the slice begins (inclusive). If omitted, it defaults to the beginning of the list.

Stop: The index where the slice ends (exclusive). If omitted, it defaults to the end of the list.

Step: The step size, which determines how many items to skip between indices. If omitted, it defaults to 1.

```
#example
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# Extracting a slice from index 2 to 5 (exclusive)
slice1 = numbers[2:6]
print(slice1)
```

```
#Example 7: Slicing with an Empty Result
# When the start index is greater than or equal to the stop index
slice11 = numbers[5:2]
print(slice11)
# When the step is negative but start is less than stop
slice12 = numbers[2:5:-1]
print(slice12)
```