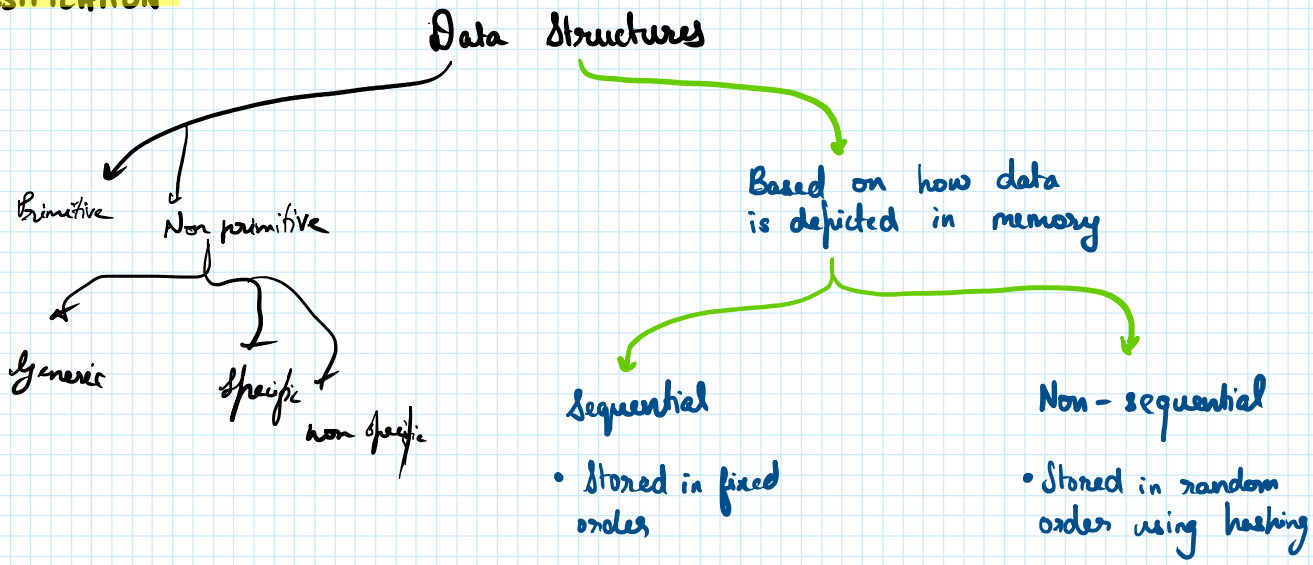
1. Data Structures - List

06 October 2023 11:38

DATA STRUCTURES

Method of storing data efficiently ⟶ data structure

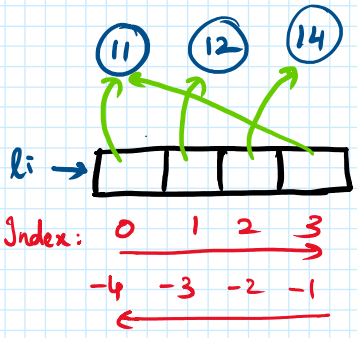
CLASSIFICATION



LIST

- Non-primitive datatype
- Sequential data structure
- Any no. of elements of any datatype
- Homogeneous / heterogeneous
  - Homogeneous → only one datatype
  - heterogeneous → more than one datatype
- Allows duplicate elements
- Indexable
- Mutable
- Iterable

li = [11, 12, 14, 11]



list slicing

- li [1::2] → [12, 11]
- li [::-1] → [11, 14, 12, 11]
- li [0:2] → [11, 12, 14]

> **NOTE:**
> If the list is empty, index does not exist. You cannot assign a value to an index that does not exist.

List functions

① append()

List functions

① append()

list_name.append([object/data to append])

Adds specified object to the end of the list. List length increased by 1.

② extend()

list_name.extend([iterable])

Takes each element one at a time from given iterable and appends it to the list. List length increased by number of elements in given iterable

③ list()

a = list([iterable])

If no argument is given ⟶ creates new empty list

If iterable argument is given ⟶ creates new list with each element of the iterable becoming an element of the list

Eg: a = list("test") ⟶ ['t', 'e', 's', 't']

list() is a constructor function.

> **NOTE: Constructor function**
> Special method in a class used to create and initialise an object of that class

④ insert()

list_name.insert(index, value)

Inserts specified value at specified index

⑤ pop()

list_name.pop(index)

Deletes last element and returns that element if no argument is specified.

If index is specified, element at that index is deleted.

⑥ remove()

list_name.remove(element)

Deletes first occurrence of given element. Only 1 argument can be specified

⑦ count()

list_name.count(value)

Returns count of occurrences of given value

⑧ reverse() / reversed()

list_name.reverse()

Reverses the original list

list_name.reverse()

Reverses the original list

reversed(list_name)

Returns reversed list without affecting original list

⑨ sort() / sorted()

list_name.sort()

Sorts list containing similar datatypes in ascending order

sorted(list_name)

Returns sorted list without affecting og. list

sorted(list_name, reverse = True)

Descending order

⑩ copy()

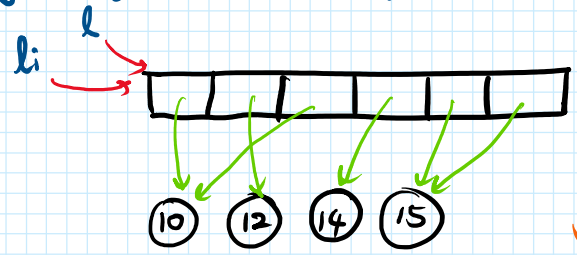
list_name.copy()

Returns a copy of specified list with a different memory location

In the following case,

li = l

Any changes in l will affect li and vice versa.

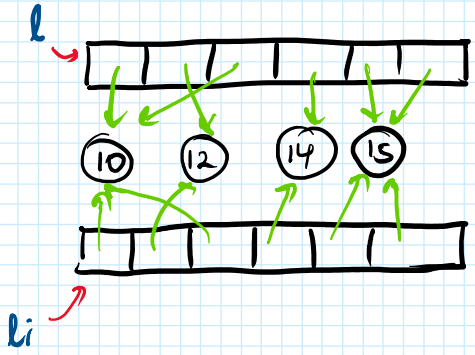


Shallow copy: No new memory location, no new id

Whereas

li = l.copy()

Changes in l does not affect li



Deep copy: new memory location, new id

NOTE:
copy() only produces a deep copy on the first level. Nested iterables inside the list are still shallow copied.
If you want to produce a true deep copy of a list:

```
import copy
li = copy.deepcopy()
```