List Data Structure in Python

Anab Batool Kazmi

Data Structures

- Data structures are organized formats or **containers** used to **store, manage, and manipulate data efficiently** within computer programs, facilitating tasks like **insertion**, **retrieval**, and **modification** of information.
- Python offers a variety of data structures, including **lists, tuples, sets, dictionaries, linked lists, trees, graphs, and more**, facilitating efficient data manipulation and storage.

Data Type Mutability in Python

```
x=10
print(id(x))
x=11
print(id(x))
```

```
1402367328
1402367360
```

Mutable Data Types in Python

- Mutable data types in Python are those whose ***values can be changed in place*** after they have been created.
- **List, Dictionary, Set**

Immutable Data Types in Python

- Immutable data types in Python are those ***whose values, once assigned, cannot be changed***, and any operation that appears to ***modify them actually creates a new object*** with the modified value.
- **Tuple, String, Numeric, Boolean**

List Data Structure

- In Python, a list is a versatile and **fundamental** data structure that allows you to store a collection of items **in an ordered sequence**. Lists are **mutable**, meaning you can modify their contents, and they can hold **elements of different data types**.
- Since lists are indexed, lists can have **duplicate values**
- Lists are created using **square brackets []** and separating items with **commas.**
- list1=[1,2,4,”a”,[7,8,9], “apple”]

Key characteristics of an ordered sequence in Python lists

- **Positional Indexing:** Each element in the list is assigned a unique position or index. The index **starts from 0 for the first element, 1 for the second element,** and so on.

- **Preservation of Order:** The order of elements in the list is preserved, meaning that **if you add elements to the list in a certain order, they will remain in that order unless you explicitly change it.**

- **Sequential Access:** You can access elements of the list in a sequential manner, starting from the **first element and moving through the list in the specified order.**

```
my_list = [10, 20, 30, 40]
```

```
my_list[0]
```

```
10
```

```
my_list[1]
```

```
20
```

```
my_list[2]
```

```
30
```

```
my_list[3]
```

Ordered sequence property

In this list:

10 is at index 0.

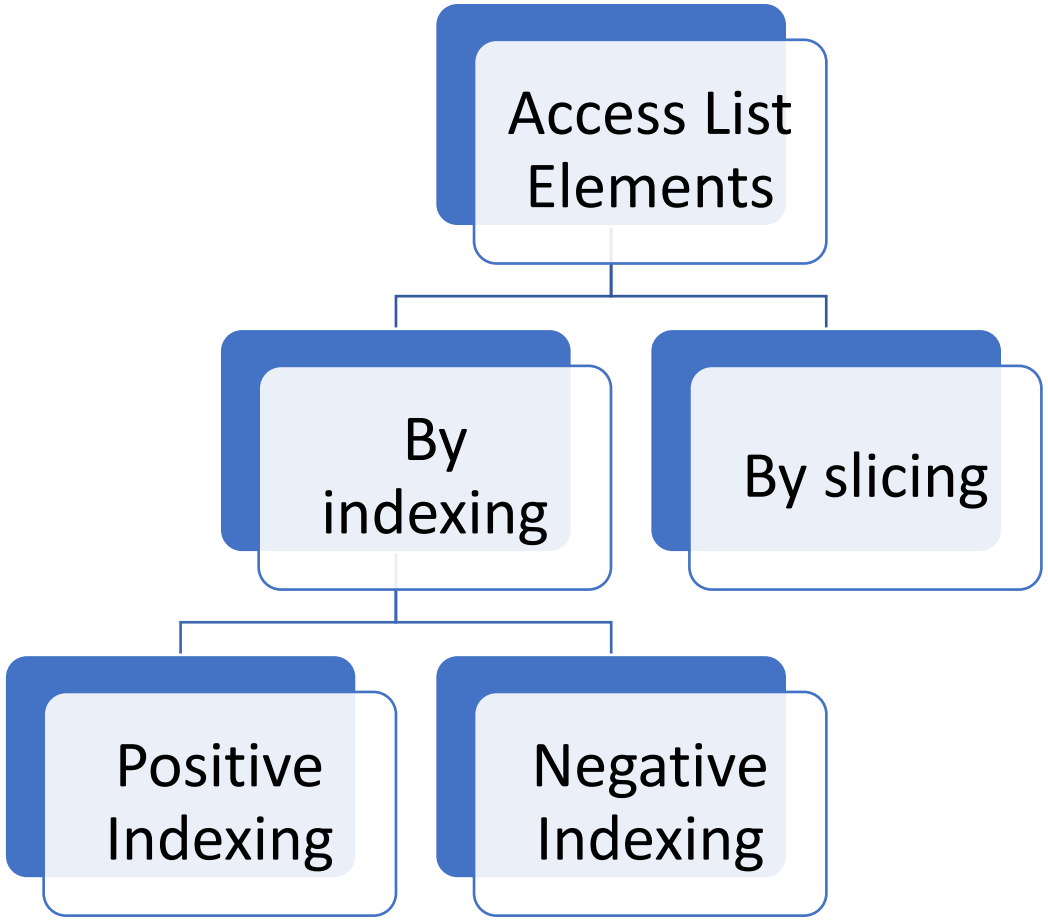
20 is at index 1.

30 is at index 2.

40 is at index 3.

The **ordered sequence property** ensures that when you iterate through the list or perform operations on it, **the elements are processed in the same order they were added.**

How to access elements of list



Indexing

- You can access a specific element in a list by providing its index inside square brackets. Indexing starts at 0 for the first element, -1 for the last element, and so on.
- countries_list=["Pakistan", "India", "Bangladesh"]

Positive indexing →

countries_list[0]	countries_list[1]	countries_list[2]
Pakistan	India	Bangladesh
countries_list[-3]	countries_list[-2]	countries_list[-1]

← Negative indexing

```
my_list = [10, 20, 30, 40]
print(my_list)
```

```
[10, 20, 30, 40]
```

```
my_list[-1]
```

```
40
```

```
my_list[-2]
```

```
30
```

```
my_list[-3]
```

```
20
```

```
my_list[-4]
```

```
10
```

Negative Indexing

- Negative indexing, **starting at -1, enables us to retrieve elements in reverse order, from the last element to the first.**

Code snippet to access array elements by Indexing

```
#accessing array elemnts by index
countries_list=["Pakistan", "India", "Bangladesh", "Iran", "Saudi Arabia"]
print(countries_list)       #print the complete list
print(countries_list[0])    # Print the first element ("Pakistan")
print(countries_list[2])    # Print the third element ("Bangladesh")
print(countries_list[-1])   # Print the last element ("Saudi Arabia")
print(countries_list[-2])   # Print the second last element ("Iran")
```

```
['Pakistan', 'India', 'Bangladesh', 'Iran', 'Saudi Arabia']
Pakistan
Bangladesh
Saudi Arabia
Iran
```

Slicing

- You can access a **range of elements** in a list using slicing.
- Slicing is done by specifying a **start index (inclusive)** and an **end index (exclusive) separated by a colon : operator**

list_name[start index : end index]

- If you **omit the start index** in slicing, it **defaults to the beginning** of the list.
- If you **omit the end index**, it defaults to the **end of the list**.

Code snippet to access array elements by Slicing

```
#accessing list elemnts by slicing
my_list = [10, 20, 30, 40, 50]
print(my_list)
sub_list = my_list[1:4]      # Access elements from index 1 to 3: [20, 30, 40]
print(sub_list)
partial_list = my_list[:3]   # Access elements from the beginning to index 2: [10, 20, 30]
print(partial_list)
```

```
[10, 20, 30, 40, 50]
[20, 30, 40]
[10, 20, 30]
```

Slicing in List with Negative Indexes

- You can also use negative indexes in slicing to specify a range of elements.

Example:

- my_list[-3:-1] retrieves a slice from the third-to-last element up to the last element, but not including, the last element.

Code snippet of Slicing in List with Negative Indexes

```
#Slicing in List with Negative Indexes
my_list = [10, 20, 30, 40, 50]
print(my_list[:])
sub_list = my_list[-4:-2]       # Access elements from index 1 to 2: [20, 30]
print(sub_list)
last_two_elements = my_list[-2:]   # Access the last two elements: [40, 50]
print(last_two_elements)
last_three_elements = my_list[-3:]   # Access the last two elements: [30, 40, 50]
print(last_three_elements)
first_two_elements = my_list[:-3]   # Access the first two elements: [10, 20]
print(first_two_elements)
```

```
[10, 20, 30, 40, 50]
[20, 30]
[40, 50]
[30, 40, 50]
[10, 20]
```

Combining Positive and Negative Indexes in Slicing

- You can mix positive and negative indexes when slicing.

```
#Slicing in List with Positive and Negative Indexes
my_list = [10, 20, 30, 40, 50]
print(my_list[1:-2])
```

```
[20, 30]
```

Combining Slicing and Indexing

- Slicing allows you to extract a range of elements from a list i.e. to extract sub list, and then access element from specific index from sub list

```
my_list = [10, 20, 30, 40, 50,60,70]
element_and_slice = my_list[0:3][-1]   # Access the second element from elements 0 to 2: 30
print(element_and_slice)
print(my_list[1:4][2]) # Access the third element from elements b/w index 1 to 3: 40
print(my_list[-5:][2]) # Access the third element from elements b/w index 2 to last: 50
```

```
30
40
50
```

Check if Item Exists in the list

- 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")
```

Python - Change List Items

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

```
# change item at specific index
thislist = ["apple", "banana", "cherry"]
print(thislist)
thislist[1] = "blackcurrant"
print(thislist)
```

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

Python - Change List Items

- 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:

```
#Change a Range of Item Values
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

Python - Change List Items

- 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"]
print(thislist)
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
['apple', 'blackcurrant', 'watermelon', 'cherry']
```

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

Python - Change List Items

- 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"]
print(thislist)
thislist[1:3] = ["watermelon"]
print(thislist)
```

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

Python - Loop Lists

- Loop Through a List items
  - using for loop
- Loop Through the Index Numbers
  - using for loop
  - using while loop

Loop through list items

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

```
#You can loop through the list items by using a for loop
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

```
apple
banana
cherry
```

Loop through list items

- 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])
```

```
apple
banana
cherry
```

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

```
apple
banana
cherry
```

How to create an Empty List

- Creating an empty list can be accomplished through two methods:
  - using square brackets []
    - name_of_list = [] #python code to generate empty list
  - employing the list() constructor.
    - name_of_list = list() #python code to generate empty list using constructor

How to create an Empty List

Empty list is a list having no elements, therefore the length of an empty list is 0, and the bool context of an empty list is false.

using square brackets []

```
import sys

# create an Empty List using square brackets []
my_list = []
print("Type of my_list data structure is :",type(my_list))
print("List Elements are :",my_list)
print("No. of Elements in my_list are :",len(my_list))
print(f"Memory reserved by my_list is {sys.getsizeof(my_list)} bytes")
print(f"Memory address of my_list is {id(my_list)}")
```

```
Type of my_list data structure is : <class 'list'>
List Elements are : []
No. of Elements in my_list are : 0
Memory reserved by my_list is 64 bytes
Memory address of my_list is 2073126281032
```

using the list() constructor

```
#create an Empty List using the list() constructor
my_list1 = list()
print("Type of my_list data structure is :",type(my_list1))
print("List Elements are :",my_list1)
print("No. of Elements in my_list are :",len(my_list1))
print(f"Memory reserved by my_list is {sys.getsizeof(my_list1)} bytes")
print(f"Memory address of my_list is {id(my_list1)}")
```

```
Type of my_list data structure is : <class 'list'>
List Elements are : []
No. of Elements in my_list are : 0
Memory reserved by my_list is 64 bytes
Memory address of my_list is 2073126252488
```

Methods to initialize a list in Python

- There are four main methods to initialize a list in Python:
  - Using square brackets
  - Using the list() constructor
  - Using list multiplication
  - Using list comprehensions

Initialize lists using the square brackets

- You can create a list by enclosing elements in square brackets.

Example:

```
my_list = [1, 2, 3, 4]
```

Initialize lists using the list() Constructor

- The **list() constructor** can be used to create a list from an **iterable** (e.g., a tuple, string, or another list).

Example:

my_list = list((1, 2, 3, 4))

```
my_list = list("hello")
print (my_list)
my_list = list((1, 2, 3, 4, 5))
print (my_list)
my_list = list([1, 2, 3, 4, 5])
print (my_list)
```

```
['h', 'e', 'l', 'l', 'o']
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

Initialize lists using the * operator in Python

- In Python, you can initialize a list by repeating its elements using the * operator.
- This is useful when you want to create a list with multiple copies of the same element.

syntax

[element] * n

where:

- **element** is the value that you want to **repeat** in the list.
- **n** is the **number of times** that you want to repeat element in the list.

Initialize lists using the * operator in Python

- You can use this technique with any element, including **numbers, strings, or even other lists**, to create a new list with repeated copies of that element.

```
#Initialize lists using the * operator in Python
list1 = [5]*10 # Creates a list with 10 copies of the element 5
print (list)
```

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

```
list2 = [False]*10 # Creates a list with 10 copies of the element False
print (list2)
```

```
[False, False, False, False, False, False, False, False, False, False]
```

```
list3 = ["Hello"]*5 # Creates a list with 5 copies of the element "Hello"
print (list3)
```

```
['Hello', 'Hello', 'Hello', 'Hello', 'Hello']
```

Using comprehensions to initialize the list in Python

- List comprehensions can be a very concise and efficient way to **initialize lists** in Python.
- They can also be used to perform more complex operations, such as **filtering and transforming** the elements of a list.

Using comprehensions to initialize the list in Python

syntax:

[expression **for** member **in** iterable **if** condition]

where:

- **expression** is the member itself, a call to a method, or any other valid expression that returns a value.
- **member** is the object or value in the list or iterable.
- **iterable** is a list, set, sequence, generator, or any other object that can return its elements one at a time.
- **condition** is an optional expression that must evaluate to True for the member to be included in the list.

Create a list of squares from 1 to 10

```
#Using comprehensions to initialize the list in Python
# Create a list of squares from 1 to 10
squares = [x * x for x in range(1, 11)]
print (squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

---

expression is x*x

Create a list of cubes from 1 to 10

```
# Create a list of cubes from 1 to 10
def cube(var):
    return var*var*var

cube_list=[cube(x) for x in range(1,11)]
print (cube_list)
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

expression is function call here.

Create a list of even numbers from 1 to 10

```python
# Create a list of even numbers from 1 to 10
even_numbers = [x for x in range(1, 11) if x % 2 == 0]
print (even_numbers)
```

```
[2, 4, 6, 8, 10]
```

Create a list of all the vowels in the string "hello"

```
# Create a list of all the vowels in the string "hello"
vowels = [x for x in "hello" if x in "aeiou"]
print (vowels)
```

```
['e', 'o']
```

Create a sub list of all the numbers between 90 and 100 from the given list

```
# Create a sub list of all the numbers between 90 and 100 from the given list
list_random=[90,10,20,50,87,90,30,50,40,35,55,60,100]
sub_list=[item for item in list_random if item>=90 and item<=100]
print(sub_list)
```

```
[90, 90, 100]
```

This list comprehension uses the **and operator to combine two conditions**.

Python - Add List Items

- Append Items
  - To add an item to the end of the list, use the **append()** method
- Insert Items
  - To insert a list item at a specified index, use the **insert()** method.
- Extend List
  - To append elements from another iterable object (list,tuples, sets, dictionaries etc.). to the current list, use the **extend()** method.

Python - Add List Items - append()

- To add an item to the end of the list, use the **append()** method
- **append()** takes exactly one argument

```
#To add an item to the end of the list, use the append() method
thislist = ["apple", "banana", "cherry"]
print(thislist)
thislist.append("orange")
print(thislist)
```

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

Python - Add List Items -insert()

- 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:

```
#To insert a new list item, without replacing any of the existing values, we can use the insert() method.
thislist = ["apple", "banana", "cherry"]
print(thislist)
thislist.insert(2, "watermelon")
print(thislist)
```

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

Python - Add List Items - extend()

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

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical) # adding list items at the end of the list
print(thislist)
thislist.extend(['KiWI']) # adding list items at the end of the list
print(thislist)
thislist.extend('KiWI') # adding string items at the end of the list
print(thislist)
thislist.extend(("t10","t20","t30")) #  adding tuple items at the end of the list
print(thislist)
```

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya', 'KiWI']
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I']
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20', 't30']
```

Python - Remove List Items

- Remove Specified Item
  - The **remove()** method removes the specified item.
  - If there are more than one item with the specified value, the **remove()** method removes the first occurrence
- Remove Specified Index
  - The **pop()** method removes the specified index.
  - If you do not specify the index, the **pop()** method removes the last item.
  - The **del** keyword also removes the specified index
- Delete list completely
  - The **del** keyword can also delete the list completely.
- Clear the list
  - The **clear()** method empties the list.
  - The list still remains, but it has no content.

Python - Remove List Items

In [117]:
```
thislist =['apple', 'banana', 'cherry', 'mango','banana', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20', 't30']
print(thislist)
thislist.remove("banana") #Remove the first occurance of "banana"
print(thislist)
thislist.pop(1) #Remove the second item i.e. item at index 1 , it will remove 'cherry'
print(thislist)
thislist.pop()  #Remove the last item , it will remove 't30'
print(thislist)
del thislist[0] # Remove the first item, i.e 'apple'
print(thislist)
thislist.clear() #Clear the list content , empty the list
print(thislist)
del thislist #Delete the entire list:
print(thislist)
```

```
['apple', 'banana', 'cherry', 'mango', 'banana', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20', 't30']
['apple', 'cherry', 'mango', 'banana', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20', 't30']
['apple', 'mango', 'banana', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20', 't30']
['apple', 'mango', 'banana', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20']
['mango', 'banana', 'pineapple', 'papaya', 'KiWI', 'K', 'i', 'W', 'I', 't10', 't20']
[]

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-117-eaef7f51902e> in <module>()
     12 print(thislist)
     13 del thislist #Delete the entire list:
---> 14 print(thislist)

NameError: name 'thislist' is not defined
```

Passing a List as an Argument to the function

- To pass a list as an argument to a function in Python, you simply pass the list to the function as you would any other argument.

```
def my_function(my_list):
  for item in my_list:
    print(item)

my_list = [1, 2, 3, 4, 5]

my_function(my_list)
```

```
1
2
3
4
5
```

Function returning list

```
def get_user_input_list():
  """Gets values from the user, appends them to a list, and returns that list.

  Returns:
    A list containing the user's input.
  """

  my_list = []
  while True:
    user_input = input("Enter a value: ")
    if user_input == "":
      break
    my_list.append(user_input)

  return my_list

my_list1 = get_user_input_list()
print(my_list1)
```

```
Enter a value: 1
Enter a value: 5
Enter a value: 7
Enter a value: 6
Enter a value:
['1', '5', '7', '6']
```

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

Reference

- https://www.w3schools.com/