1.List

Definition: In Python, a**list**is a built-in dynamic sized array (automatically grows and shrinks). We can store all types of items (including another list) in a list. A list may contain mixed type of items, this is possible because a list mainly stores references at contiguous locations and actual items maybe stored at different locations.

* List can contain duplicate items.
* List in Python are Mutable. Hence, we can modify, replace or delete the items.
* List are ordered. It maintain the order of elements based on how they are added.
* Accessing items in List can be done directly using their position (index), starting from 0.

Example:

a = [10, 20, 15]

print(a[0]) *# access first item*

a.append(11) *# add item*

a.remove(20) *# remove item*

print(a)

**Output**

10

[10, 15, 11]

USES:

1.Lists are used to store multiple item in a single variable.

2.Lists are one of 4 built-in data types in python used to store collection of data.

Declaration:

In Python, a list is declared by enclosing a comma-separated collection of items in square brackets ([]). Here are some examples of declaring a list:

1. **Empty list**:

my\_list = []

2. **List with integers**:

my\_list = [1, 2, 3, 4, 5]

3. **List with different data types**:

1. my\_list = [1, "hello", 3.14, True]

**4.List of lists (nested lists)**:

my\_list = [[1, 2], [3, 4], [5, 6]]

**Accessing List Elements**

Elements in a list can be accessed using **indexing**. Python indexes start at **0**, so **a[0]** will access the first element, while**negative indexing** allows us to access elements from the end of the list. Like index -1 represents the last elements of list.

a = [10, 20, 30, 40, 50] *# Access first element*

print(a[0]) *# Access last element*

print(a[-1])

**Output:**

**10**

**50**

**Updating Elements into List**

We can change the value of an element by accessing it using its index.

a = [10, 20, 30, 40, 50] *# Change the second element*

a[1] = 25

print(a)

**Output**

[10, 25, 30, 40, 50]

**Removing Elements from List**

We can remove elements from a list using:

* [**remove()**](https://www.geeksforgeeks.org/python-list-remove/)**:**Removes the first occurrence of an element.
* [**pop()**](https://www.geeksforgeeks.org/python-list-pop-method/)**:**Removes the element at a specific index or the last element if no index is specified.
* [**del statement**](https://www.geeksforgeeks.org/python-del-to-delete-objects/)**:** Deletes an element at a specified index.

Example:

a = [10, 20, 30, 40, 50]

a.remove(30)

print("After remove(30):", a)

popped\_val = a.pop(1) *# Removes the element at index 1 (20)*

print("Popped element:", popped\_val)

print("After pop(1):", a) *# Deletes the first element (10)*

**del** a[0]

print("After del a[0]:", a)

**Output**

After remove(30): [10, 20, 40, 50]

Popped element: 20

After pop(1): [10, 40, 50]

After del a[0]: [40, 50]

Operation on list:

In Python, you can perform several operations on lists. Here's a breakdown of some common operations:

**1. Accessing Elements**

* Access individual elements by their index (starting from 0).

Example:

my\_list = [10, 20, 30, 40]

print(my\_list[1])

Output: 20

**2. Modifying Elements**

* You can change the value of a list element using its index.

Example:

my\_list[2] = 35

print(my\_list) # Output: [10, 20, 35, 40]

**3. Appending an Element**

* You can add an element to the end of the list using append().

Example:

my\_list.append(50)

print(my\_list) # Output: [10, 20, 35, 40, 50]

**4. Inserting an Element**

* Use insert() to add an element at a specific position.

Example:

my\_list.insert(1, 15) # Inserts 15 at index 1

print(my\_list) # Output: [10, 15, 20, 35, 40, 50]

**5. Removing Elements**

* Remove an element by value using remove().

Example:

my\_list.remove(20) # Removes the first occurrence of 20

print(my\_list) # Output: [10, 15, 35, 40, 50]

* You can also remove an element by index using pop().

Example:

my\_list.pop(2) # Removes the element at index 2 (35)

print(my\_list) # Output: [10, 15, 40, 50]

**6. List Concatenation**

* Use + to concatenate lists.

Example:

list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined\_list = list1 + list2

print(combined\_list) # Output: [1, 2, 3, 4, 5, 6]

**7. Repeating a List**

* Use \* to repeat the list multiple times.

Example:

my\_list = [1, 2, 3]

repeated\_list = my\_list \* 2

print(repeated\_list) # Output: [1, 2, 3, 1, 2, 3]

**8. Slicing Lists**

* You can slice lists to extract a portion of the list.

Example:

my\_list = [10, 20, 30, 40, 50]

sliced\_list = my\_list[1:4] # Extracts from index 1 to 3 (not including 4)

print(sliced\_list) # Output: [20, 30, 40]

**9. Finding Length**

* Use len() to get the length of the list.

Example:

my\_list = [10, 20, 30, 40, 50]

print(len(my\_list)) # Output: 5

**10. Checking if an Element Exists**

* Use in to check if an element is in the list.

Example:

my\_list = [10, 20, 30, 40]

print(20 in my\_list) # Output: True

print(25 in my\_list) # Output: False

**11. Sorting a List**

* Use sort() to sort the list in ascending order (modifies the list in place).

Example:

my\_list = [40, 10, 30, 20]

my\_list.sort()

print(my\_list) # Output: [10, 20, 30, 40]

* Use sorted() to get a sorted copy of the list (without modifying the original).

sorted\_list = sorted(my\_list)

print(sorted\_list) # Output: [10, 20, 30, 40]

**12. Reversing a List**

* Use reverse() to reverse the list in place.

Example:

my\_list.reverse()

print(my\_list) # Output: [40, 30, 20, 10]

**13. Finding an Index**

* Use index() to find the index of an element.

Example:

my\_list = [10, 20, 30, 40]

print(my\_list.index(30)) # Output: 2

**14. List Comprehension**

* You can create a new list using list comprehension.

Example:

my\_list = [1, 2, 3, 4, 5]

squares = [x \*\* 2 for x in my\_list] # Square each element

print(squares) # Output: [1, 4, 9, 16, 25]

**15. Extending a List**

* Use extend() to append elements from another list to the end.

Example:

my\_list = [1, 2, 3]

my\_list.extend([4, 5, 6])

print(my\_list)

Output: [1, 2, 3, 4, 5, 6]

Top of Form

Bottom of Form

: Write a programme to demonstrate various operations to access the elements of a list.

CODE:

num\_list = [1, 3, 5, 40, 50, 60, 100, 70]

print("1st element of the list:", num\_list[0]) print("num\_list[2:5]:", num\_list[2:5])

print(len(num\_list))

num\_list1 = [1, 2, 3, 4, 5, 7]

list3 = num\_list + num\_list1

print(len(list3))

list3.insert(4, 500)

print(list3)

list3.append(400)

print(list3)

del list3[2]

print(list3)

del list3[1:4]

print(list3)

print(max(list3))

print(sorted(list3))

print(min(list3))

print(list(reversed(list3)))

OUTPUT:

1st element of the list: 1

num\_list[2:5]: [5, 40, 50]

8

14

[1, 3, 5, 40, 500, 50, 60, 100, 70, 1, 2, 3, 4, 5, 7]

[1, 3, 5, 40, 500, 50, 60, 100, 70, 1, 2, 3, 4, 5, 7, 400]

[1, 3, 40, 500, 50, 60, 100, 70, 1, 2, 3, 4, 5, 7, 400]

[1, 500, 50, 60, 100, 70, 1, 2, 3, 4, 5, 7, 400]

500

[1, 2, 3, 4, 5, 7, 50, 60, 70, 100, 400, 500]

1

[500, 400, 100, 70, 60, 50, 7, 5, 4, 3, 2, 1]

2B-TUPLE:

Definition: Python Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

*# Note : In case of list, we use square*

*# brackets []. Here we use round brackets ()*

t = (10, 20, 30)

print(t)

print(type(t))

**Output**

(10, 20, 30)

<class 'tuple'>

Uses:

* When you need to store a collection of values that shouldn't be changed.
* When you need to ensure the integrity of the data.
* When you need to use the collection as a key in a dictionary.
* When performance (memory and speed) is a concern for fixed-size collections.

Declaration:

In Python, a tuple is an immutable, ordered collection of elements. It is declared using parentheses () or the tuple() constructor. Here are some ways to declare a tuple:

1. Using Parentheses ()

# Empty tuple

empty\_tuple = ()

# Tuple with elements

num\_tuple = (1, 2, 3, 4, 5)

str\_tuple = ("apple", "banana", "cherry")

# Tuple with mixed data types

mixed\_tuple = (1, "hello", 3.14, True)

2. Using tuple() Constructor

list\_to\_tuple = tuple([1, 2, 3]) # Converts list to tuple

string\_to\_tuple = tuple("hello") # Converts string to tuple

3. Single Element Tuple (Comma is Necessary)

single\_element\_tuple = (42,) # Must include a trailing comma

not\_a\_tuple = (42) # This is just an integer, NOT a tuple

4. Nested Tuples

nested\_tuple = ((1, 2, 3), ("a", "b", "c"))

5. Tuple Packing and Unpacking

# Packing

packed\_tuple = 1, 2, 3 # Parentheses are optional

# Unpacking

a, b, c = packed\_tuple # a=1, b=2, c=3

**Different Operations Related to Tuples**

Below are the different operations related to tuples in Python:

**Traversing Items of Python Tuples**

Like List Traversal, we can traverse through a tuple using for loop.

*# Define a tuple*

t = (1, 2, 3, 4, 5)

*# Traverse through each item in the tuple*

**for** x **in** t:

print(x, end=" ")

**Output**

1 2 3 4 5

**Concatenation of Python Tuples**

To [Concatenation](https://www.geeksforgeeks.org/python-string-concatenation/) of Python Tuples, we will use plus operators(+).

*# Code for concatenating 2 tuples*

t1 = (0, 1, 2, 3)

t2 = ('python', 'geek')

*# Concatenating above two*

print(t1 + t2)

**Output**

(0, 1, 2, 3, 'python', 'geek')

**Nesting of Python Tuples**

A nested tuple in Python means a tuple inside another tuple.

*# Code for creating nested tuples*

t1 = (0, 1, 2, 3)

t2 = ('python', 'geek')

t3 = (t1, t2)

print(t3)

**Output**

((0, 1, 2, 3), ('python', 'geek'))

**Repetition Python Tuples**

We can create a tuple of multiple same elements from a single element in that tuple.

*# Code to create a tuple with repetition*

t = ('python',)\*3

print(t)

**Output**

('python', 'python', 'python')

**Slicing Tuples in Python**

Slicing a Python tuple means dividing a tuple into small tuples using the indexing method. In this example, we slice the tuple from index 1 to the last element. In the second print statement, we printed the tuple using reverse indexing. And in the third print statement, we printed the elements from index 2 to 4.

*# code to test slicing*

t = (0 ,1, 2, 3)

print(t[1:])

print(t[::-1])

print(t[2:4])

**Output**

(1, 2, 3)

(3, 2, 1, 0)

(2, 3)

**Note:**In Python slicing, the end index provided is not included.

**Deleting a Tuple in Python**

In this example, we are deleting a tuple using ‘[del’ keyword](https://www.geeksforgeeks.org/python-del-to-delete-objects/). The output will be in the form of error because after deleting the tuple, it will give a NameError.

**Note:**Remove individual tuple elements is not possible, but we can delete the whole Tuple using Del keyword.

*# Code for deleting a tuple*

t = ( 0, 1)

**del** t

print(t)

**Output:**

Hangup (SIGHUP)  
Traceback (most recent call last):  
 File "Solution.py", line 5, in <module>  
 print(t)  
NameError: name 't' is not defined

**Finding the Length of a Python Tuple**

To find the length of a tuple, we can use Python’s len() function and pass the tuple as the parameter.

*# Code for printing the length of a tuple*

t = ('python', 'geek')

print(len(t))

**Output**

2

**Multiple Data Types With Tuple**

Tuples in Python are heterogeneous in nature. This means tuples support elements with multiple datatypes.

*# tuple with different datatypes*

t = ("immutable", **True**, 23)

print(t)

**Output**

('immutable', True, 23)

**Converting a List to a Tuple**

We can convert a [list](https://www.geeksforgeeks.org/python-lists/) in Python to a tuple by using the tuple() constructor and passing the list as its parameters.

*# Code for converting a list and a string into a tuple*

a = [0, 1, 2]

t = tuple(a)

print(t)

**Output**

(0, 1, 2)

**Output:**

Tuples take a single parameter which may be a list, string, set, or even a dictionary(only keys are taken as elements), and converts them to a tuple.

**Tuples in a Loop**

We can also create a tuple with a single element in it using [loops](https://www.geeksforgeeks.org/loops-in-python/).

*# python code for creating tuples in a loop*

t = ('gfg',)

*# Number of time loop runs*

n = 5

**for** i **in** range(int(n)):

t = (t,)

print(t)

**Output**

(('gfg',),)

((('gfg',),),)

(((('gfg',),),),)

((((('gfg',),),),),)

(((((('gfg',),),),),),)