Page No.
Date: | |

13/02/2025

☆ Python List -
In a python, a list is a built-in dynamic sized array (automaticlly grows and shrinks). we can store all type of items, this is possible because a list mainly stores references at contiguous location 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 maintains 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.

```
   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]
```

Date : | |

★ Creating a List ÷

1. using square Brackets ÷

```
# List of integers
  a = [1, 2, 3, 4, 5]

# List of strings
  b = ['apple', 'banana', 'cherry']

# mixed data types
  c = [1, 'hello', 3.14, True]

print (a)
print (b)
print (c)
```

OUTPUT :

```
[1, 2, 3, 4, 5]
['apple', 'banana', 'cherry']
[1, 'hello', 3.14, True]
```

★ Using list () Constructor

We can also create a list by passing an iterable (like a string, tuple, or another list) to list () function.

```
# From a tuple
  a = list((1,2,3, 'apple', 4.5))
  print (a)
```

OUTPUT!

```
[1, 2, 3, 'apple', 4.5]
```

Page No.
Date: | |

* Creating List with repeated Elements:
-> We can create a list with repeated ~~set~~ elements using the multiplication operator.

```
# Create a list [2,2,2,2,2]
  a = [2] * 5    => [2,2,2,2,2]

# create a list [0,0,0]
  b = [0] * 3  => [0, 0, 0]

print (a)
print (b)
```

* 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
```

Page No.
Date: | |

★ Adding Elements into list ÷

We can add elements to a list using the following methods:

• append (): add an element at the end of the list

• extend (): add multiple elements to the end of the list.

• insert (): add an element at specific position.

```
# initialize an empty List
  a = []

# adding 10 to end of list
 a · append (10)
 print ("After append (10) :" a)

# Inserting 5 at index 0
 a. insert (0, 5)
 print (" after insert (0, 5): ", a)

# adding multiple elements [15, 20, 25]
     at the end

 a. extend ([15, 20, 25])
  print ("after extend ([15, 20, 25]); ", a)
```

OUTPUT

```
·  After append (10) : [10]
   after insert (0, 5) : [5, 10]
   after extend [[15, 20, 25]]: [5, 10, 15, 20, 25]
```

Page No.
Date: | |

☆ 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)
```

☆ Removing Elements from list:
We can remove elements from a list using:

1, remove () : Removes the first occurence of an element.

2, pop () : Remove the element at a specific index or the element if no idex is specified.

3, del statement: Deletes an element at a specified index.

```
a= [10, 20, 30, 40, 50]
# Removes the first occurrence of 30.
 a. remove (30)
 print ("after remove (30: ", a)
```

=> popped_val = a. pop (1)

=> del a[0]

★ Iterating over lists
we can iterate the lists easily by using a for loop or other iteration methods. Iterating over lists is useful when we want to do some operation on each item or access specific items based on certain conditions.

- using for loop:

```
a = ['apple', 'banana', 'cherry'].
# Iterating over the list.
for item in a:
        print(item)
```

OUTPUT:

```
apple
banana
cherry
```

★ Nested lists in python:
A nested list is a list within another list, which is useful for representing matrices or tables. we can access nested elements by chaining indices.

```
matrix =[
         [1, 2, 3],
         [4, 5, 6]
         [7, 8, 9].
       ]
print (matrix [1] [2])
  6
```

Page No.
Date: | |

✪ Python arrays

List in python are the most flexible and commonly used data structure for sequential storage, they are similar to arrays in other languages but with several key differences:

• Dynamic Typing: python lists can hold elements of different types in the same list.

• Dynamic Resizing: Lists are dynamically resized, meaning you can add or remove elements without declaring the size of the list upfront.

• Built-in methods: Python lists come with numerous built-in methods that allow for easy manipulation of the elements within them, including methods for appending removing, sorting and reversing element.

Example:

```
a= [1, "Hello", [3.14, "World"]]
a.append (2)
Print(a)
```

Page No.
Date: / /

✯ Python Tuples:

- A tuple in Python is an immutable ordered collection of elements. Tuples are simmilar to list but unlike lists, they cannot be changed after their creation (i.e.. they are immutable).

- Tuples can hold elements of different data types. the main characteristics of tuples are being ordered, heterogenious and immutable.

★ Creating a Tuple:

- A tuple is created by placing all the items inside parentheses (), separated by commas. A tuple can have any number of items and they can be different data Types.

Example:

```
# Creating an empty Tuple
  tup = ()
  print (tup)

# using string
  tup =( 'Geeks', 'For')
  print (tup)
```

```
# using list
 li =  [1,2,4,5,6]
print (tuple (li))

# Using Built-in function
  tup = tuple ('Geeks')
  print (tup)
```

OUTPUT

```
 ()
 ('Geeks', 'For')
 (1,2,4,5,6)
 ('G', 'e', 'e', 'k', 's')
```

⇨ Creating a Tuple with mixed Datatypes.
- Tuple can contain elements of various data types, including other tuples, list, dictionaries and eve functions.

- Example:

```
  # Creating a Tuple with mixed Datatype:
  tup = (5, 'Welcome', 7, 'Geeks')
  print (tup)

  # Creating a Tuple with nested tuples.
  tup 1 = (0, 1, 2, 3)
  tup 2 = ('Python', 'geek')
  tup 3 = (tup 1, tup 2)
  print (tup3)
```

Page No.
Date : | |

Creating a Tuple with repetition
```
tup1 = ('Geeks',)*3
print(tup1)
```

Creating a Tuple with repetition the use of Loop.
```
tup = ('Geeks')
n = 5
for i in range (int(n)):
    tup = (tup,)
    Print (tup)
```

OUTPUL :-
```
(5, 'Welcome', 7, 'Geeks')
((0, 1, 2, 3), ('Python', 'geek'))
('Geeks', 'Geeks', 'Geeks')
('Geeks',)
(('Geeks',),)
((('Geeks',),),)
(((('Geeks',),),),)
((((('Geeks',),),),),)
```

Page No.
Date: | |

☆ Python Tuple operations

-> Accessing of Tuples:
• we can access the elements of a tuple by using indexing and slicing.

• Indexing starts at 0 for the first element and goes up to n-1, where n is the number of elements in the tuple.

• Negative indexing start from -1 for the last element and goes backward.

• Example:
```
# Accessing the tuple with Indexing
tup = tuple("Geeks")
print (tup [0])

# Accessing a range of elements using slicing
print (tup [1: 4])
print (tup [: 3])

# Tuple unpacking
tup = ("Geeks", "For", "Geeks")

# This line unpack values of Tuple 1
a, b, c = tup
print (a)
print (b)
print (c)
```

OUTPUT :-

```
G
('e', 'e', 'k')
('G', 'e', 'e')
Geeks
For
Geeks
```

-> Concatenation of Tuples:

Tuples can be concatenated using the + operator. This operation combines two or more tuples to create a new tuple.

NOTE: Only the same datatypes can be ~~Con~~ Combined with concatenation, an error arises if a list and a tuple are combined.

Tuple 1

0	1	2	3

Tuple 2

'Su	Po	lu

Concatenated Tuple

0	1	2	3	Su	Po	lu

```
tup1 = (0, 1, 2, 3)
tup2 = ('Geeks', 'For', 'Geeks')

~~tup3 = ('Geeks', 'For', 'Geeks')~~
tup3 = tup1 + tup2
print (tup3)
```

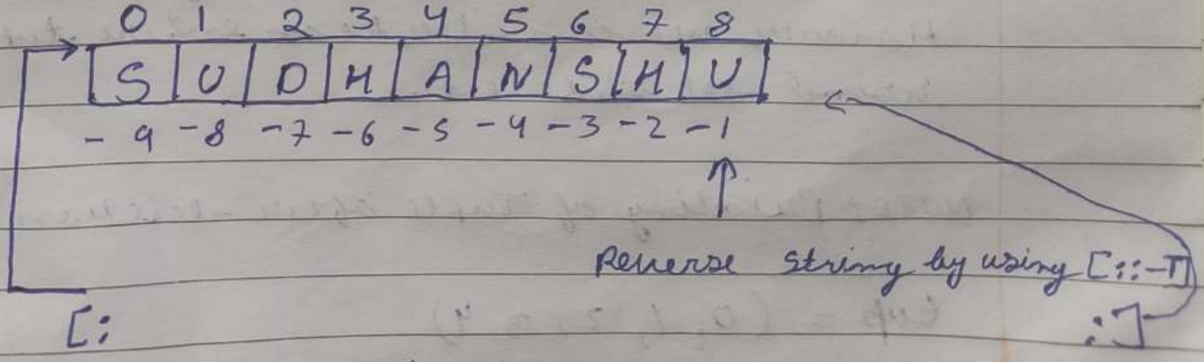
Output
```
(0, 1, 2, 3, 'Geeks', 'For, 'Geeks')
```

→ Slicing of Tuple :

- Slicing a Tuple means creating a new tuple from a subset of elements of the original tuple.

- the slicing syntax is tuple [start:stop: step].

Note : Negative increment values can also be used to reverse the sequence of tuples.



Default Beginning of sequence

Default end of the sequence

Slicing of a Tuple with numbers:

```
tup = tuple('SUDHANSHU')
# Removing First element
print (tup [1:])

# Reversing the tup
print ( tup[::-1])

# Printing elements of a Range
Print (tup [4:9]
```

OUTPUT :

```
('U','D','H','A','N','S','H','U')

('U','H','S','N','A','H','D','U','S')

('A','N','S','H')
```

-> Deleting a Tuple
Since tuple are immutable, we cannot delete individual elements of a tuple. However we can delete an entire tuple using del statement.

NOTE: Printing of Tuple after deletion result in Error

```
tup = (0, 1, 2, 3, 4)
del tup
print (tup)
```

Page No.
Date: | |

· => Tuple

* Tuple Built-in methods:

Tuple support only a few methods due to their immutable nature. The two most commonly used methods are Count () and index ()

- index() = Find in the tuple and returns the index of the given value where it's available.

- Count () = Returns the frequency of a specified value.

* Tuple Built-In Functions:

Functions:

- all (). => Return true if all element are true or if tuple is empty.

- any (). => return true if any element of the tuple is true. if tuple is empty, return false.

- len (). => return length of the tuple or size of the tuple.

- ~~enumerate(). Return length of the tuple or size of the tuple.~~

- enumerate (). Returns enumerated object of tuple.

- max (). return maximum element of given tuple.

- min (). return ~~maximum~~ minimum element of given tuple.

- Sum (). Sum up the numbers

- Sorted (). input elements in the tuple and return a new sorted list.

- tuple (). Convert an iterable to a tuple