List Data Type in Python

DATA STRUCTURES IN PYTHON:-

Data Structures

Data Structures allow us to store and organize data efficiently. This will allow us to easily access and perform operations on the data.

In Python, there are four built-in data structures

- *List*
- *Tuple*
- *Set*
- *Dictionary*

DISCUSSION ON LIST DATA TYPE:-

List is the most versatile python data structure. Holds an ordered sequence of items.

Creating a List:-

Created by enclosing elements within [square] brackets. Each item is separated by a comma.

Code example:-

```
a = 2list_a = [5, "Six", a, 8.2]
print(type(list_a))
print(list_a)
```

Output:-

```
<class 'list'>
[5, 'Six', a,8.2]
```

OPERATIONS ON LISTS:-

1. Creating a List of Lists

Code example:-

```
a = 2
list_a = [5, "Six", a, 8.2]
list_b = [1, list_a]
print(list_b)
```

Output:-

```
[1, [5, 'Six', 2, 8.2]]
```

Operations on lists:-

Length of a List

Code example:-

```
a = 2
list_a = [5, "Six", a, 8.2]
print(len(list_a))
```

Output:-4

Accessing List Items

To access elements of a list, we use Indexing.

code example:-

```
a = 2
list_a = [5, "Six", a, 8.2]
print(list_a[1])
```

output:- Six

Operations on lists:-

Iterating Over a List:-

code example:-

```
a = 2
list_a = [5, "Six", a, 8.2]
for item in list_a:
    print(item)
```

Output:-

```
5
Six
a
8.2
```

Operations on lists:-

List Concatenation

Similar to strings, + operator concatenates lists.

Code Example:-

```
list_a = [1, 2, 3]

list_b = ["a", "b", "c"]

list_c = list_a + list_b

print(list_c)
```

Output:-

[1, 2, 3, 'a', ‘b’, ’c’]

Operations on Lists:-

Adding Items to List

Code Example:-

```
list_a = []
print(list_a)
for i in range(1,4):
    list_a += [i]
print(list_a)
```

Output:-

```
[]
[1, 2,3]
```

Operations on Lists:-

Repetition

* Operator repeats lists.

Code Example:-

```
list_a = [1, 2]
list_b = list_a * 3
print(list_b)
```

Output:-

[1,2,1,2,1,2]

Operations on Lists:-

List Slicing

Obtaining a part of a list is called List Slicing.

Code Example:-

```
list_a = [5, "Six", 2, 8.2]
list_b = list_a[:2]
print(list_b)
```

Output:-

```
[5,'Six']
```

Operations on Lists:-

Extended Slicing

Similar to string extended slicing, we can extract alternate items using step.

Code Example:-

```
list_a = ["R", "B", "G", "O", "W"]
list_b = list_a[0:5:3]
print(list_b)
```

Output:-

['R', ‘O’]

Operations on Lists:-

Converting to List

list(sequence) takes a sequence and converts it into list.

Code Example:-

```
color = "Red“
list_a = list(color)
print(list_a)
```

Output:-

['R', 'e', ‘d’]

Example 2:-

```
list_a = list(range(4))
print(list_a)
```

Output:-[0,1,2,3]

Operations on Lists:-

Lists are Mutable

- Lists can be modified.Items at any position can be updated.
- Code example:-

```
list_a = [1, 2, 3, 5]
print(list_a)
list_a[3] = 4
print(list_a)
```

Output:-

```
[1,2,3,5]
[1,2,3,4]
```

Note:-

Strings are Immutable

Strings are Immutable (Can’t be modified).

Code Example:-

```
message = "sea you soon“
message[2] = "e“
print(message)
```

TypeError: 'str' object does not support it

Working With Lists:-

Object:-

In general, anything that can be assigned to a variable in Python is referred to as an object.

Strings, Integers, Floats, Lists etc. are all objects.

Examples

- "A"
- 1.25
- [1,2,3]

Identity of an Object

Whenever an object is created in Python, it will be given a unique identifier (id).

This unique id can be different for each time you run the program.

Every object that you use in a Python Program will be stored in Computer Memory.

The unique id will be related to the location where the object is stored in the Computer Memory.

Working With Lists:-

Id of Lists

Code Example:-

```
list_a = [1, 2, 3]
list_b = [1, 2, 3]
print(id(list_a))
print(id(list_b))
```

Output:-

```
139637858236800
139637857505984
```

Working With Lists:-

Modifying Lists - 1

When assigned an existing list both the variables list_a and list_b will be referring to the same object.

Code Example:-

```
list_a = [1, 2, 3]
list_b = list_a
print(id(list_a))
print(id(list_b))
```

Output:-

```
140334087715264
140334087715264
```

Working With Lists:-

When assigned an existing list both the variables list_a and list_b will be referring to the same object.

Code Example:-

```
list_a = [1, 2, 3, 5]
list_b = list_a
list_b[3] = 4
print("list a : " + str(list_a))
print("list b : " + str(list_b))
```

Output:-

```
list a : [1, 2, 3, 4]
list b : [1, 2, 3, 4]
```

Working With Lists:-

Modifying Lists - 3

The assignment will update the reference to new object.

Code Example:-

```
list_a = [1, 2]
list_b = list_a
list_a = [6, 7]
print("list a : " + str(list_a))
print("list b : " + str(list_b))
```

Output:-

```
list a : [6, 7]
list b : [1, 2]
```

Working With Lists:-

Modifying Lists - 4

The assignment will update the reference to a new object.

Code Example:-

```
list_a = [1, 2]
list_b = list_a
list_a = list_a + [6, 7]
print("list a : " + str(list_a))
print("list b : " + str(list_b))
```

Output:-

```
list a : [1, 2, 6, 7]
list b : [1,2]
```

Working With Lists:-

Modifying Lists - 5

Compound assignment will update the existing list instead of creating a new object.

Code Example:-

```
list_a = [1, 2]
list_b = list_a
list_a += [6, 7]
print("list a : " + str(list_a))
print("list b : " + str(list_b))
```

Output:-

```
list a : [1, 2, 6, 7]
list b : [1, 2, 6, 7]
```

Working With Lists:-

Modifying Lists - 6

Updating mutable objects will also effect the values in the list, as the reference is changed.

Code Example:-

```
list_a = [1,2]
list_b = [3, list_a]
list_a[1] = 4
print(list_a)
print(list_b)
```

Output:-

```
[1, 4]
[3, [1, 4]]
```

Working With Lists:-

Modifying Lists - 7

Updating immutable objects will not effect the values in the list, as the reference will be changed.

Code Example:-

```
a = 2
list_a = [1,a]
print(list_a)
a = 3
print(list_a)
```

Output:-

```
[1, 2]
[1, 2]
```

Lists and Strings

Splitting

str_var.split(separator)

Splits a string into a list at every specified separator. If no separator is specified, default separator is whitespace.

Code Example:-

```
nums = "1 2 3 4“

num_list = nums.split()

print(num_list)
```

Output:-

```
['1', '2', '3’, ‘4’]
```

More on Strings and lists:-

Multiple WhiteSpaces

Multiple whitespaces are considered as single when splitting.

Code Example:-

```
nums = "1    2 3 4 “
num_list = nums.split()
print(num_list)
```

Output:-

```
['1', '2', '3', '4’]
```

New line \n and tab space \t are also whitespace.

Code Example:-

```
nums = "1\n2\t3 4“
num_list = nums.split()
print(num_list)
```

Output:- ['1', '2', '3', '4']

More on Strings and lists:-

Using Separator

Breaks up a string at the specified separator.

Code Example 1:-

```
nums = "1,2,3,4“
num_list = nums.split(',’)
print(num_list)
```

Output:- ['1', '2', '3', '4’]

Example 2:

```
nums = "1,2,,3,4,“
num_list = nums.split(',’)
print(num_list)
```

Output:-['1', '2', '', '3', '4', '']

More on Strings and lists:-

Space as Separator

Code Example:-

```
nums = "1  2 3 4 “
num_list = nums.split(" ")
print(num_list)
```

Output:-

```
['1', '', '2', '3', '4', ‘’]
```

More on Strings and lists:-

String as Separator

Example – 1:-

```
string_a = "Python is a programming language“
list_a = string_a.split('a’)
print(list_a)
```

Output:- ['Python is ', ' progr', 'mming l', 'ngu', 'ge’]

Example 2:-

```
string_a = "step-by-step execution of code“
list_a = string_a.split('step’)
print(list_a)
```

Output:- ['', '-by-', ' execution of code']

More on Strings and lists:-

Joining

str.join(sequence)

Takes all the items in a sequence of strings and joins them into one string.

Code Example:-

```
list_a = ['Python is ', ' progr', 'mming l', 'ngu', 'ge’]

string_a = "a".join(list_a)

print(string_a)
```

Output:-Python is a programming language

Note:-

Joining Non String Values

Sequence should not contain any non-string values.

Code Example:-

```
list_a = list(range(4))
string_a = ",".join(list_a)
print(string_a)
```

Output:-

```
TypeError: sequence item 0: expected str instance, int found
```

Negative Indexing:-

Using a negative index returns the nth item from the end of list.

Last item in the list can be accessed with index -1.

Reversing a List

-1 for step will reverse the order of items in the list.

Code Example:-

```
list_a = [5, 4, 3, 2, 1]

list_b = list_a[::-1]

print(list_b)
```

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

Accessing List Items

Code Example 1:-

```
list_a = [5, 4, 3, 2, 1]
item = list_a[-1]
print(item)
```

Output: 1

Example 2:-

```
list_a = [5, 4, 3, 2, 1]
item = list_a[-4]
print(item)
```

Output:- 4

Slicing With Negative Index

```
list_a = [5, 4, 3, 2, 1]
 list_b = list_a[-3:-1]
 print(list_b)
```

Output: - [3,2]

Out of Bounds Index

While slicing, Index can go out of bounds.

Example:-

```
list_a = [5, 4, 3, 2, 1]
list_b = list_a[-6:-2]
print(list_b)
```

Output:- [5,4,3]

Negative Step Size

variable[start:end:negative_step]

Negative Step determines the decrement between each index for slicing.

Start index should be greater than the end index in this case

- start > end
- Code Example:-list_a = [5, 4, 3, 2, 1]

```
list_b = list_a[4:2:-1]
print(list_b)
```

Output:- [1, 2]

Example - 2

Negative step requires the start to be greater than end.

Code Example:-

```
list_a = [5, 4, 3, 2, 1]

list_b = list_a[2:4:-1]

print(list_b)
```

Output:- []

Reversing a List

-1 for step will reverse the order of items in the list.

Code Example:-

```
list_a = [5, 4, 3, 2, 1]

list_b = list_a[::-1]

print(list_b)
```

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

Note:-

Negative Step Size - Strings

Code Examples:-

```
string_1 = "Program“
string_2 = string_1[6:0:-2]
print(string_2)
```

Output:- mro

```
string_1 = "Program“
string_2 = string_1[-4:-1]
print(string_2)
```

Ouput:- gra