**Data Structures**

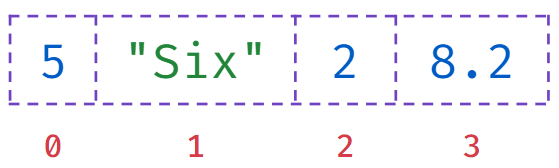
Data Structures allow us to store and organize data efficiently. This will allow us to easily access and perform operations on the data.

In Python, there are four built-in data structures

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

List

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**



1

2

3

4

a = 2

list\_a = [5, "Six", a, 8.2]

print(type(list\_a))

print(list\_a)

PYTHON

**Output**



<class 'list'>

[5, 'Six', 2, 8.2]

Creating a List of Lists

**Code**



1

2

3

4

a = 2

list\_a = [5, "Six", a, 8.2]

list\_b = [1, list\_a]

print(list\_b)

PYTHON

**Output**



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

Length of a List

**Code**



1

2

3

a = 2

list\_a = [5, "Six", a, 8.2]

print(len(list\_a))

PYTHON

**Output**



4

Accessing List Items

To access elements of a list, we use Indexing.

**Code**



1

2

3

a = 2

list\_a = [5, "Six", a, 8.2]

print(list\_a[1])

PYTHON

**Output**



Six

Iterating Over a List

**Code**



1

2

3

4

a = 2

list\_a = [5, "Six", a, 8.2]

for item in list\_a:

print(item)

PYTHON

**Output**



5

Six

2

8.2

List Concatenation

Similar to strings,

+

operator concatenates lists.

**Code**



1

2

3

4

list\_a = [1, 2, 3]

list\_b = ["a", "b", "c"]

list\_c = list\_a + list\_b

print(list\_c)

PYTHON

**Output**



[1, 2, 3, 'a', 'b', 'c']

Adding Items to List

**Code**



1

2

3

4

5

list\_a = []

print(list\_a)

for i in range(1,4):

list\_a += [i]

print(list\_a)

PYTHON

**Output**



[]

[1, 2, 3]

**Repetition**

\*

Operator repeats lists.

**Code**



1

2

3

list\_a = [1, 2]

list\_b = list\_a \* 3

print(list\_b)

PYTHON

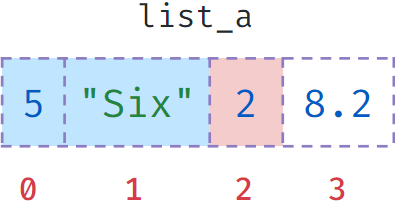
**Output**



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

List Slicing

Obtaining a part of a list is called List Slicing.



**Code**



1

2

3

list\_a = [5, "Six", 2, 8.2]

list\_b = list\_a[:2]

print(list\_b)

PYTHON

**Output**



[5, 'Six']

Extended Slicing

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

**Code**



1

2

3

list\_a = ["R", "B", "G", "O", "W"]

list\_b = list\_a[0:5:3]

print(list\_b)

PYTHON

**Output**



['R', 'O']

Converting to List

list(sequence)

takes a sequence and converts it into list.

**Code**



1

2

3

color = "Red"

list\_a = list(color)

print(list\_a)

PYTHON

**Output**



['R', 'e', 'd']

**Code**



1

2

list\_a = list(range(4))

print(list\_a)

PYTHON

**Output**



[0, 1, 2, 3]

Lists are Mutable

* Lists can be modified.
* Items at any position can be updated.

**Code**



1

2

3

4

list\_a = [1, 2, 3, 5]

print(list\_a)

list\_a[3] = 4

print(list\_a)

PYTHON

**Output**



[1, 2, 3, 5]

[1, 2, 3, 4]

Strings are Immutable

Strings are Immutable (Can’t be modified).

**Code**



1

2

3

message = "sea you soon"

message[2] = "e"

print(message)

PYTHON

**Output**



TypeError: 'str' object does not support item assignment

Notes

S

**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]

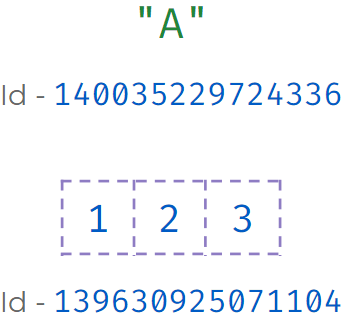
Identity of an Object

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.



Finding Id

We can use the

id()

to find the id of a object.

**Code**



1

print(id("Hello"))

PYTHON

**Output**



140589419285168

Id of Lists



1

2

3

4

list\_a = [1, 2, 3]

list\_b = [1, 2, 3]

print(id(list\_a))

print(id(list\_b))

PYTHON

**Output**



139637858236800

139637857505984

Modifying Lists

Modifying Lists - 1

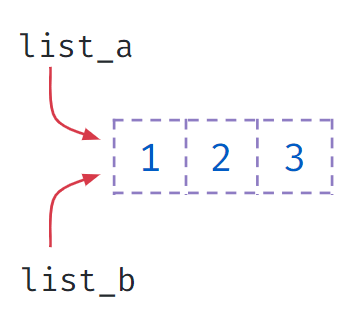
When assigned an existing list both the variables

list\_a

and

list\_b

will be referring to the same object.



**Code**



1

2

3

4

list\_a = [1, 2, 3]

list\_b = list\_a

print(id(list\_a))

print(id(list\_b))

PYTHON

**Output**



140334087715264

140334087715264

Modifying Lists - 2

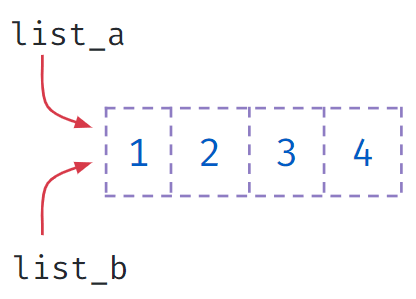
When assigned an existing list both the variables

list\_a

and

list\_b

will be referring to the same object.



**Code**



1

2

3

4

5

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))

PYTHON

**Output**

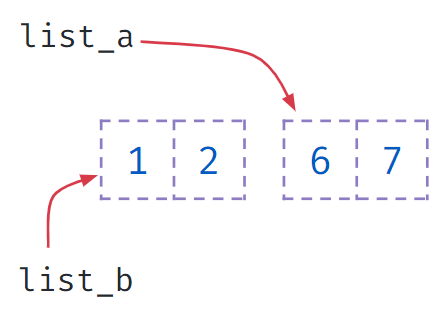


list a : [1, 2, 3, 4]

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

Modifying Lists - 3

The assignment will update the reference to new object.



**Code**



1

2

3

4

5

list\_a = [1, 2]

list\_b = list\_a

list\_a = [6, 7]

print("list a : " + str(list\_a))

print("list b : " + str(list\_b))

PYTHON

**Output**

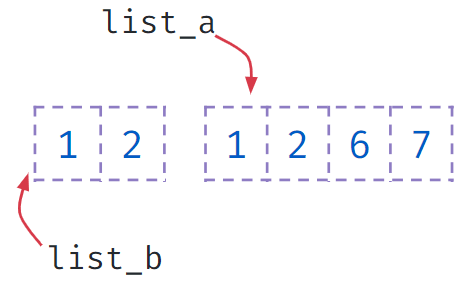


list a : [6, 7]

list b : [1, 2]

Modifying Lists - 4

The assignment will update the reference to a new object.



**Code**



1

2

3

4

5

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))

PYTHON

**Output**

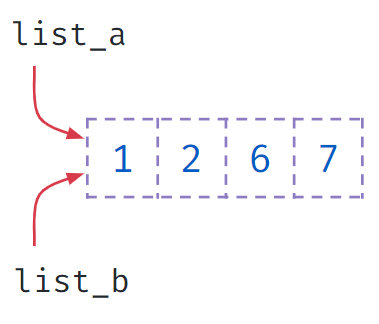


list a : [1, 2, 6, 7]

list b : [1, 2]

Modifying Lists - 5

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



**Code**



1

2

3

4

5

list\_a = [1, 2]

list\_b = list\_a

list\_a += [6, 7]

print("list a : " + str(list\_a))

print("list b : " + str(list\_b))

PYTHON

**Output**

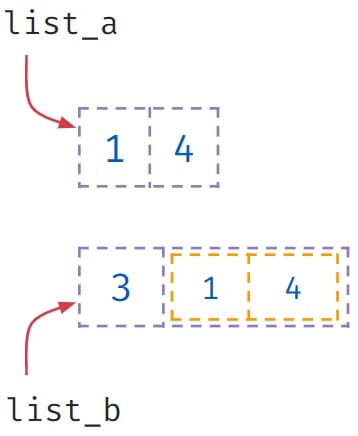


list a : [1, 2, 6, 7]

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

Modifying Lists - 6

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



**Code**



1

2

3

4

5

list\_a = [1,2]

list\_b = [3, list\_a]

list\_a[1] = 4

print(list\_a)

print(list\_b)

PYTHON

**Output**

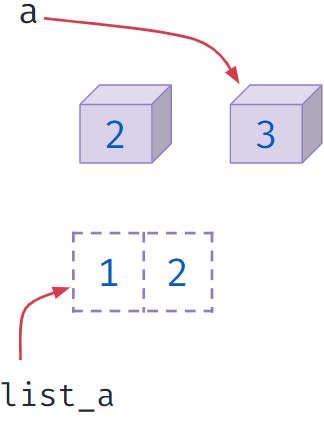


[1, 4]

[3, [1, 4]]

Modifying Lists - 7

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



**Code**



1

2

3

4

5

a = 2

list\_a = [1,a]

print(list\_a)

a = 3

print(list\_a)

PYTHON

**Output**