**Sequences**

A sequence is not a data type; it’s a **category of data types**.

Here are several types of sequences in Python; the following three are the most important.

Lists are the most versatile sequence type. The elements of a list can be any object, and lists are mutable - they can be changed. Elements can be reassigned or removed, and new elements can be inserted.

For example:

my\_name = "Jack"

hello = "Hello World"

my\_age = 27

my\_list = [my\_name, my\_age, hello]

Tuples are like lists, but they are immutable - they can’t be changed.  
*Note*: Tuples use parenthesis

my\_tuple = (1+3, 2.7, 'Thursday')

Strings are a particular type of sequence that can only store characters, and they have a special notation. However, all of the sequence operations described below can also be used on strings.

my\_string = "I love cheese!"

**Indexing Sequences**

Sequences can be indexable, which means each element can be retrieved with his index : sequence\_name[index\_num].

For example, using a list

my\_name = "Jack"

hello = "Hello World"

my\_age = 27

my\_list = [my\_name, hello, my\_age]

print(my\_list[0]) # The result gives us "Jack"

You may wonder why does my\_list[0] give me the first item in my list? Be sure to take note that indexes start from [0], which would be the first item in the list, and index [1], will give us the second item in our list being the hello variable with the value of "Hello World".

There are other indexing methods:

* Negative indexes, meaning starting from the end to the beginning (sequence\_name[-index\_num]) this is an example using a tuple:

my\_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9)

print(my\_tuple[-2]) # The result gives us the number 8

* Range indexes return a list from one index to the other (sequence\_name[start:end]), this is an example using a list:

my\_list = [10, 20, 30, 40, 50, 60, 70, 80, 90]

print(my\_list[0:4]) # The result gives us [10, 20, 30, 40]

When using ranges, the end index is **not included**

**Strings**

Actually, str is a sequence; it’s a sequence of letters.  
You can index the letters of a string with the indexing technique.

>>> my\_name = "Rick"

>>> print(my\_name[0])

"R"

>>> print(my\_name[2])

"c"

>>> print(my\_name[-1])

"k"

>>> print(my\_name[1:3])

"ic"

**Lists**

Lists let you store a collection of items. These items can be of any type: integers, strings, booleans or even other lists.  
See the official documentation [**here**](https://docs.python.org/3/tutorial/datastructures.html#more-on-lists)

A List is a collection that is ordered and changeable. It allows duplicate members.

To create a list, use square brackets [], and insert elements, separated by comma ,. Elements can be every data type.

For example:

>>> my\_hobbies = ["Eat", "Sleep", "Code"]

To access the items, refer to the index number.

>>> my\_hobbies[0]

"Eat"

>>> my\_hobbies[2]

"Code"

>>> my\_hobbies[-2]

"Sleep"

* **Modify an element**

The list is mutable; it means that the contents of a list can be updated. To change an element in a list, refer to the index number and assign a new value.

**Careful:**Indexes in list start from 0, not from 1

>>> print(my\_hobbies)

["Eat", "Sleep", "Code"]

>>> my\_hobbies[1] = "Meditate"

>>> print(my\_hobbies)

["Eat", "Meditate", "Code"]

Be careful! If you try to refer to an index that doesn’t exist, your program will crash to an IndexError

* **Add an element**

Adding an element to a list is called appending.

To append an item to a list, use the append() method.

>>> print(my\_hobbies)

["Eat", "Meditate", "Code"]

>>> my\_hobbies.append("Baseball")

>>> print(my\_hobbies)

["Eat", "Meditate", "Code", "Baseball"]

* **Remove an element**

There are several methods to remove items from a list.

* To remove a specified item, use the remove method; this method removes the first occurrence of the element.

>>> print(my\_hobbies)

["Eat", "Meditate", "Code"]

>>> my\_hobbies.remove("Meditate")

>>> print(my\_hobbies)

["Eat", "Code"]

* To remove an item by its index, use the pop method; this method removes the element at the given index and returns it.

>>> print(my\_hobbies)

["Eat", "Code"]

>>> my\_hobbies.pop(0)

>>> print(my\_hobbies)

["Code"]

* **Adding two lists**

Adding two lists will concatenate them.

>>> my\_hobbies = ["Food", "Code"]

>>> additional\_hobbies = ["Sport", "More code"]

>>> my\_hobbies + additional\_hobbies

["Food", "Code", "Sport", "More code"]

**List Functions**

List have several built-in functions you can use.

* **Length**  
  Using the len() function allows to retrieve the number of items contained in a given list.

fruits = ["apple","pear", "banana", "melon"]

len(fruits)

>> 4

* **Sum**  
  If you have a list of numbers, you can get the sum of all numbers again through a function.

numbers = [3, 12, 1, -4]

sum(numbers)

>> 12

* **Sorting**  
  There are different ways of sorting elements of a list, sorted() lets you straightforwardly do that.

If you have numbers, a list will be sorted from smallest to biggest.

numbers = [3, 5, 1, 2]

sorted(numbers)

>> [1, 2, 3, 5]

If you have strings, they will get sorted alphabetically.

letters = ['d', 'a', 'g', 'b']

sorted(letters)

>> ['a', 'b', 'd', 'g']

It will not work if you have mixed values.

mixed = [3, 'd', 1, 'a']

sorted(mixed)

>> error!

**Tuples**

Tuples are immutable lists, which means items **can’t be changed**.

To create a tuple, use parentheses:

>>> my\_tuple = (5,6,7)

The good thing about tuples is that they can be unpacked, meaning each value will go to one variable:

>>> a, b, c = my\_tuple

>>> print(a)

5

>>> print(b)

6

>>> print(c)

7

**Sets**

**Important!!**

Lists and tuples are standard Python data types that store values in a sequence.

**Sets and dictionaries have no notion of an element index**, and therefore CANNOT be considered sequences. In mathematics, informally speaking, a sequence is an ordered list of objects (or events). Like a set, it contains members (also called elements, or terms).

Sets are another standard Python data type that also store values. The major difference is that sets, unlike lists or tuples :

* cannot have multiple occurrences of the same element
* and store unordered values

We use Sets to make a list with no duplicates in it.

To create a set, use the set() function

To add an element to a set, use set.add(element)

>>> my\_set = set()

>>> my\_set.add("Rick")

>>> my\_set.add("Morty")

>>> my\_set.add("Rick")

>>> print(my\_set)

{"Rick", "Morty"}

>>> this\_set = {"banana", "apple", "cherry", "apple"}

>>> print(this\_set)

{'banana', 'cherry', 'apple'}

# the set named this\_set is unordered