

Digital Career Institute

Python Course - Collections



Collections in Python

Collections in Python

DLI

LINEAR

LIST

TUPLE

ASSOCIATIVE

DICTIONARY

SET

Lists

Collections in Python

Lists are Python's basic implementation of linear collections.

They:

- Have an **order**.
- **Allow duplicate** values.
- Allow their objects to be **changed**.
- Allow objects of **different types**.

Defining Lists

They are defined using square brackets `[]`.

Any **iterable** can be converted into a list by using the `list` constructor.

The `list` constructor can also be used to create empty lists if no argument is given.

The output is the same as using empty square brackets `[]`.

```
>>> fridge = [  
...     "Apple", "Apple",  
...     "Cabbage", "Steak",  
...     "Cheese", "Apple",  
...     "Carrot", "Carrot",  
...     "Iogurt", "Beer"  
... ]  
>>> hello = list("hello")  
>>> print(hello)  
['h', 'e', 'l', 'l', 'o']  
>>> empty_fridge = list()  
>>> print(empty_fridge)  
[]
```

Defining Lists

Lists can contain values of any type.

Items in the list can be of mixed types.

The items themselves can also be lists.

Lists of integers can be generated using `range(start, end, step)`.

```
>>> fridge = ["Apple", "Apple"]
>>> letters = list("hello")
>>> ages = [32, 45, 42, 12, 34, 57]
>>> dates = [datetime, datetime]
>>> data = ["John", 32, datetime]
>>> lists = [
...     ["John", "Mary", "Amy"],
...     [32, 43, 51]
... ]
>>> sequence = range(2, 10, 3)
>>> print(sequence)
[2, 5, 8]
```


Python Lists: Accessing Values

DLI

Printing the list will show its values in the exact **same order** used when the list was defined.

Each value in the list can be accessed using its numeric **position** in the list (starting at zero). This is named the **index**.

Slicing can be used to access parts of the list or reverse the order of the list.



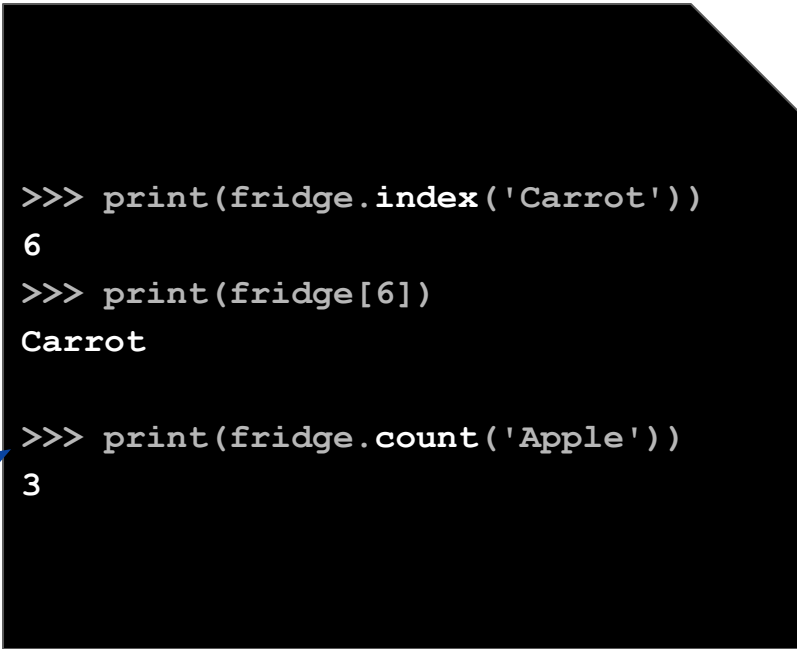
```
>>> print(fridge)
['Apple', 'Apple', 'Cabbage', ...]
>>> print(fridge[0])
Apple
>>> print(fridge[4])
Cheese
>>> print(fridge[0:2])
['Apple', 'Apple']
>>> print(fridge[-2:])
['Iogurt', 'Beer']
>>> print(fridge[::-1])
['Beer', 'Iogurt', 'Carrot', ...]
```


Python Lists: Accessing Values

Using indexes to access or slice lists is useful when we know them, but this is often not the case and a mechanism to search the values must be available.

This can be done with the method `index`, that returns the index of the first occurrence in the list.

The number of occurrences of a value can be obtained with the `count` method.



```
>>> print(fridge.index('Carrot'))
6
>>> print(fridge[6])
Carrot

>>> print(fridge.count('Apple'))
3
```

Python Lists: Order

DLI

The order of the list can be reversed with slicing if we don't want to make the change permanent.



```
>>> print(fridge[::-1])  
['Beer', 'Iogurt', 'Carrot', ...]  
>>> print(fridge)  
['Apple', 'Apple', 'Cabbage', ...]
```

The method **reverse** will make the change permanent on the original variable name and will return **None**.



```
>>> print(fridge.reverse())  
None  
>>> print(fridge)  
['Beer', 'Iogurt', 'Carrot', ...]
```

Python Lists: Sorting

DLI

The list can also be sorted ascending or descending. The `sort` method can be used to change the order of the items according to their values.



```
>>> print(fridge.sort())
None
>>> print(fridge)
['Apple', 'Apple', 'Apple', ...]
```

The argument `reverse` can be used to reverse the order of the sorting.



```
>>> fridge.sort(reverse=True)
>>> print(fridge)
['Steak', 'Iogurt', 'Cheese', ...]
>>> fridge.sort()
>>> fridge.reverse()
>>> print(fridge)
['Steak', 'Iogurt', 'Cheese', ...]
```

This is a shortcut for using first `sort()` and then `reverse()`.



Python Lists: Changing Values

DLI

The values in the collection can be changed by using indexes.

A set of contiguous values can be changed in a single instruction as well.

Setting a value to **None** will not remove the value from the list.

Setting a value to an empty list will not remove the value either.



```
>>> fridge[0] = 'Beer'
>>> print(fridge)
['Beer', 'Apple', 'Cabbage',...]
>>> fridge[0:2] = ['Juice']
>>> print(fridge)
['Juice', 'Cabbage', 'Steak',...]
>>> fridge[0] = None
>>> print(fridge)
[None, 'Cabbage', 'Steak',...]
>>> fridge[0] = []
>>> print(fridge)
[[], 'Cabbage', 'Steak',...]
```

Python Lists: Adding Values

Adding new values can be done with the method **append**. Values are added at the end.



Adding new values using **append** does not change the index of the other values.

Two or more lists can also be concatenated using the **+** operator. This is equivalent to using the **extend** method.



```
>>> fridge.append('Soda')
>>> print(fridge[-2:])
['Beer', 'Soda']

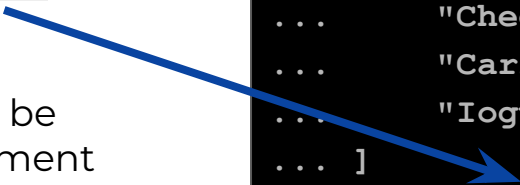
>>> eggs = ['Egg', 'Egg']
>>> fridge = fridge + eggs
>>> print(fridge[-4:])
['Beer', 'Soda', 'Egg', 'Egg']

>>> fridge = fridge.extend(eggs)
>>> print(fridge[-4:])
['Egg', 'Egg', 'Egg', 'Egg']
```

Python Lists: Adding Values

New values can be added also in other positions of the list with the `insert` method.

The position where the value will be inserted is indicated as first argument and then the value to be inserted as second argument.



```
>>> fridge = [  
...     "Apple", "Apple",  
...     "Cabbage", "Steak",  
...     "Cheese", "Apple",  
...     "Carrot", "Carrot",  
...     "Iogurt", "Beer"  
... ]  
>>> fridge.insert(1, 'Pie')  
>>> print(fridge)  
['Apple', 'Pie', 'Apple', ...]
```

Python Lists: Removing Values

DLI

Removing values can be done with the method **pop**. Values are removed from the end and returned, which can then be assigned to another variable name.

The **pop** method accepts an argument that will be the index of the value to be removed and returned.

The **remove** method finds the first occurrence of the given value and removes it from the list. It returns **None**.

```
>>> last_item = fridge.pop()
>>> print(fridge[-2:])
['Carrot', 'Iogurt']
>>> print(last_item)
Beer

>>> first_item = fridge.pop(0)
>>> print(fridge)
['Pie', 'Apple', 'Cabbage',...]

>>> print(fridge.remove('Apple'))
None
>>> print(fridge)
['Pie', 'Cabbage', 'Steak',...]
```

Comparing Python Lists

Comparing two lists will return **True** if the following statements are true for the elements in it:

- They are the same
- They are in the same order

```
>>> list1 = [1, 2, 3]
>>> list2 = [1, 2, 3]
>>> list1 == list2
True
>>> list1 is list2
False
>>> list3 = [3, 2, 1]
>>> list1 == list3
False
>>> list1 == list3[::-1]
True
>>> list4 = [1, 1, 2, 2, 3, 3]
>>> list1 == list4
False
```


Python Lists: Use Case Examples

DLI

FRIDGE

- Apple
- Carrot
- logurt
- Apple
- logurt
- Beer
- Steak
- Cabbage
- Eggplant
- Orange Juice

SENSOR DATA

For instance, temperature.

- 15
- 15
- 16
- 17
- 19
- 20
- 22
- 22
- 20
- 21

GOLD MEDALS

- M. Mayer
- D. Hermann
- J. Ludwig
- Z.W. Ren
- Z.W. Ren
- U. Bogataj
- U. Bogataj
- A. Fontana
- J.T. Boe
- B. Karl

Python List Methods: Summary

Lists have a variety of methods:

Add

- Append
- Extend
- Insert

Remove

- Pop
- Remove

Search & Analyze

- Index
- Count

Sort

- Sort
- Reverse

```
>>> print(dir(fridge))  
[..., 'append', 'count', 'extend',  
'index', 'insert', 'pop', 'remove',  
'reverse', 'sort']
```

Tuples

Collections in Python

Tuples are like lists but they cannot be changed.

They are the equivalent of constants for collections and are faster than lists.

They:

- Have an **order**.
- **Allow duplicate** values.
- **Do not allow** their objects to be **changed**.
- Allow objects of **different types**.

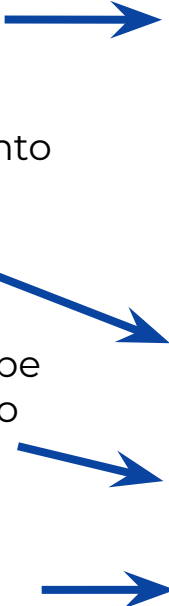
Defining Tuples

They are defined using parentheses `()`.

Any **iterable** can be converted into a tuple by using the **tuple** constructor.

The **tuple** constructor can also be used to create empty tuples if no argument is given.

The output is the same as using empty parentheses `()`.



```
>>> days = (  
...     "Monday",  
...     "Tuesday",  
...     "Wednesday",  
...     "Thursday",  
...     "Friday",  
... )  
>>> hello = tuple("hello")  
>>> print(hello)  
('h', 'e', 'l', 'l', 'o')  
>>> empty_tuple = tuple()  
>>> print(empty_tuple)  
()
```

Defining Tuples

Tuples can contain values of any type.



Items in the tuple can be of mixed types.



The items themselves can also be tuples.



```
>>> fridge = ("Apple", "Apple")
>>> letters = tuple("hello")
>>> ages = (32, 45, 42, 12, 34, 57)
>>> dates = (datetime, datetime)
>>> data = ("John", 32, datetime)
>>> tuples = (
...     ("John", "Mary", "Amy"),
...     (32, 43, 51)
... ]
```

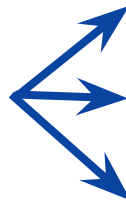
Python Tuples: Accessing Values

DLI

Printing the tuple will show its values in the exact **same order** used when the tuple was defined.

Each value in the tuple can be accessed using its numeric **position** in the tuple (starting at zero). This is named the **index**.

Slicing can be used to access parts of the tuple or reverse its order.



```
>>> print(days)
('Monday', 'Tuesday', ...)
>>> print(days[0])
Monday
>>> print(days[4])
Friday
>>> print(days[0:2])
('Monday', 'Tuesday')
>>> print(days[-2:])
('Thursday', 'Friday')
>>> print(days[::-1])
('Friday', 'Thursday', ...)
```

Python Tuples: Accessing Values

DLI

Tuples also have the method **index** implemented to return the index of the first occurrence of the given value in the tuple.

The number of occurrences of a value can also be obtained with the **count** method.



```
>>> print(days.index('Thursday'))  
3  
>>> print(days[3])  
Thursday
```

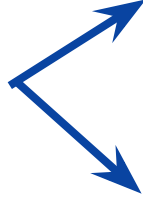


```
>>> print(days.count('Monday'))  
1
```


Python Tuples: Changing Values

DCI

The items inside a tuple cannot be changed.



```
>>> days[0] = "Sunday"
Traceback (most recent call last):
  File "/home/DCI/test.py", line 71,
in <module>
    days[0] = "Sunday"
TypeError: 'tuple' object does not
support item assignment
```

If an item in the tuple is another type of iterable, its contents can still be changed.



```
>>> days = (["Monday"], ["Tuesday"])
>>> days[0][0] = "Sunday"
>>> days[1].pop()
>>> days[1].append("Monday")
>>> print(days)
(['Sunday'], ['Monday'])
```

Python Tuples: Sort, Add & Remove

As opposed to lists, tuples cannot be changed.

Therefore, **they do not have** any of the methods that can be used in lists to manipulate its contents:

- Append
- Extend
- Insert
- Pop
- Remove
- Sort
- Reverse

```
>>> days.append("Saturday")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> days.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'pop'
>>> days.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
```

Comparing Python Tuples

Comparing two tuples will return **True** if the following statements are true for the elements in it:

- They are the same
- They are in the same order

```
>>> list1 = (1, 2, 3)
>>> list2 = (1, 2, 3)
>>> list1 == list2
True
>>> list1 is list2
False
>>> list3 = (3, 2, 1)
>>> list1 == list3
False
>>> list1 == list3[::-1]
True
>>> list4 = [1, 1, 2, 2, 3, 3]
>>> list1 == list4
False
```

Python Tuples: Use Case Examples

DLI

WEEK DAYS

- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday
- Sunday

MONTHS

- January
- February
- March
- April
- May
- June
- July
- August
- September
- October
- November
- December

MEDAL TYPES

- Gold
- Silver
- Bronze

Python Tuple Methods: Summary

DLI

Tuples only have methods to search and analyze the values:

- Index
- Count

```
>>> print(dir(days))  
[..., 'count', 'index']
```