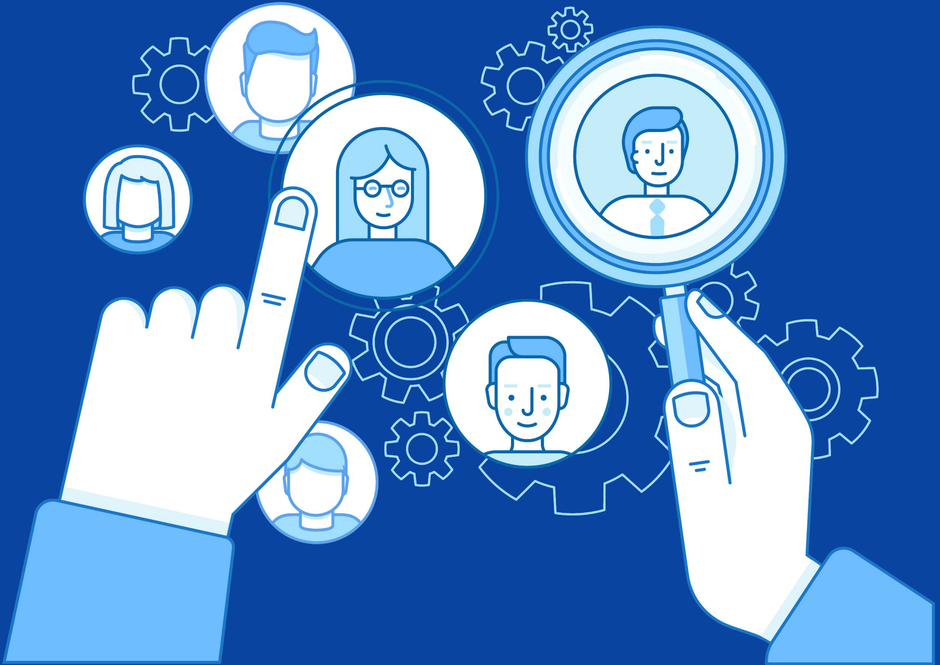
Digital Career Institute

Python Course - Collections



Digital Career Institute

DCI

Goal of the Submodule

The goal of this submodule is to help the learners work with collections in Python. By the end of this submodule, the learners should be able to understand

- What is a collection
- Which are the built-in collection types in Python
- The characteristics and differences between each collection type.
- How to choose the right data type for each situation.
- The additional data types provided by the `collections` Python module.
- How to use counters, ordered dictionaries, chain maps and named tuples.

Digital Career Institute

Topics

- Collections and iterables.
  - Types of collections: linear, associative and graphs.
  - Linear collections: types and I/O methods.
  - Associative collections: sets and arrays.
- Lists.
- Tuples.
- Sets.
- Dictionaries.
- Operations with iterables.
  - Iterating iterables.
  - Iterable functions.
- The `collections` Python module.
  - Counters.
  - Ordered dictionaries.
  - Chain maps.
  - Named tuples.

Digital Career Institute

Collections

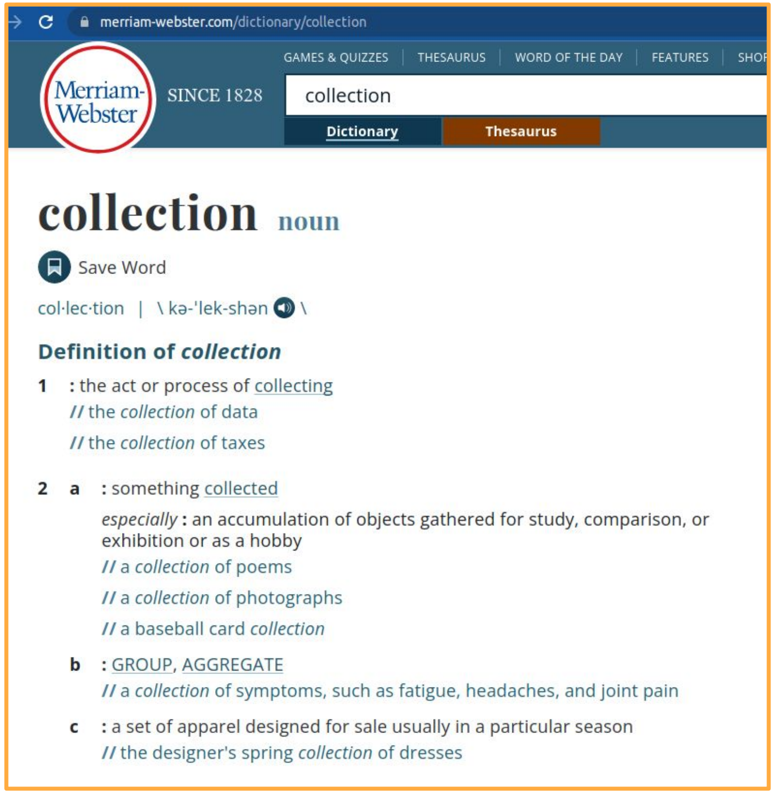
Digital Career Institute

What is a Collection?

Digital Career Institute

Source: https://unsplash.com/photos/JG35CpZLfVs

What is a Collection?



collection noun

col·lec·tion | \ kə-ˈlek-shən \

Definition of *collection*

1 : the act or process of collecting
// the *collection* of data
// the *collection* of taxes

2 a : something collected
especially **:** an accumulation of objects gathered for study, comparison, or exhibition or as a hobby
// a *collection* of poems
// a *collection* of photographs
// a baseball card *collection*

b : GROUP, AGGREGATE
// a *collection* of symptoms, such as fatigue, headaches, and joint pain

c : a set of apparel designed for sale usually in a particular season
// the designer's spring *collection* of dresses

An **accumulation** of objects.

A **group** of objects.

An **aggregate** of objects.

What is a Collection?

Digital Career Institute

A collection of **groceries**

A collection of **computers**

A collection of **customers**

What is a Collection?

H	e	l	l	o		W	o	r	l	d

A text string is also a collection, of characters.

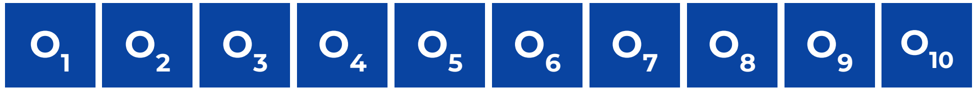
What is a Collection?

In computer programming a collection is a type of **iterable**.

An iterable is any type, which value can be decomposed into different members, and its members can be returned one at a time (i.e. in `for` loops).

Storing Collections: Linear

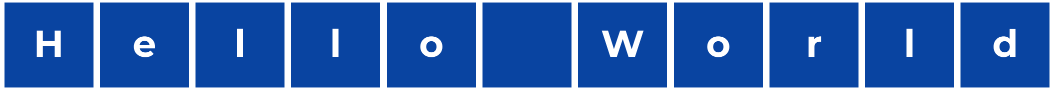
Objects of a collection can be stored in a line, one after the other.



These are called **Linear Collections**.

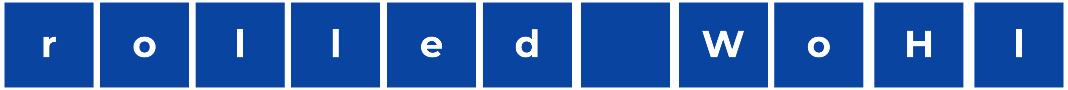
Examples:
- A queue of customers.
- A list of students.
- The steps in a recipe.
- The names of one’s children.

Storing Linear Collections



Placing the objects in line implicitly defines an order.

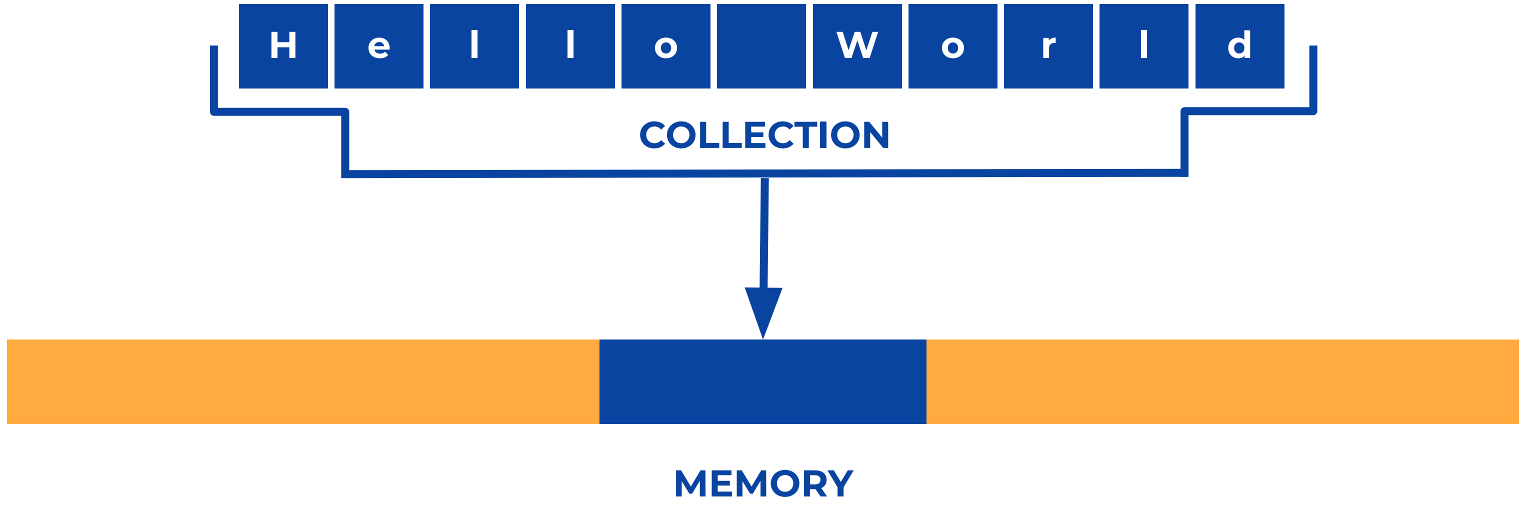
A different order of the same elements is a different sequence.



A linear collection is called
a **sequence** In Python.

Storing Linear Collections

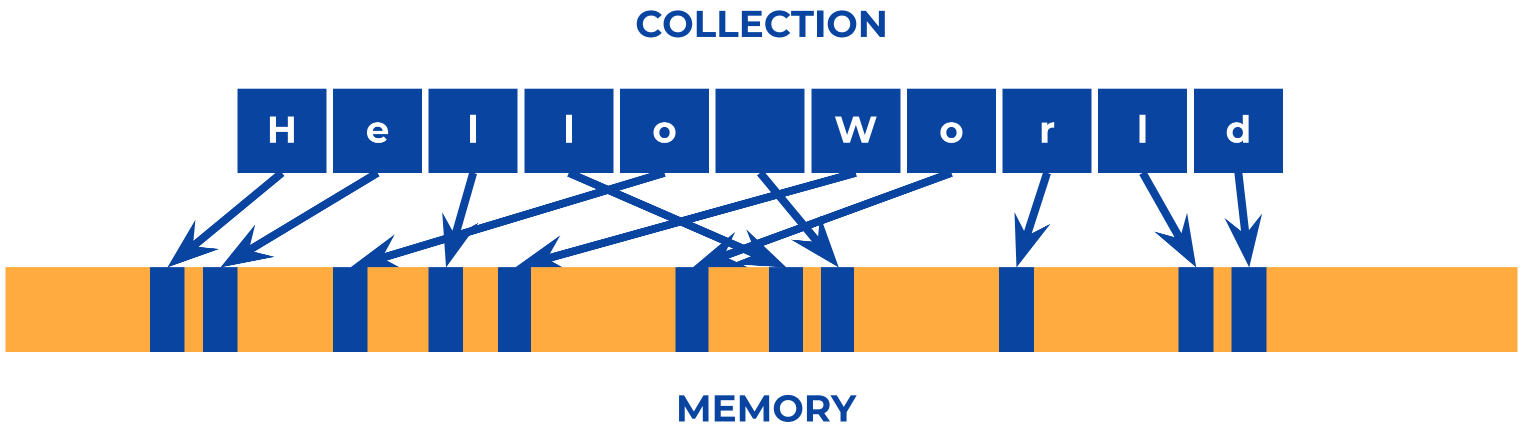
To identify the order, the objects are often stored using contiguous spaces in memory. Then, the order of the sequence is **implicit** in memory.



Most language data types use this approach.

Storing Linear Collections

Items can also be stored in different locations in memory.
Then, the order of the sequence must be made **explicit**.



To identify the order, each item will hold its value
and a link to the next item in the sequence.
These are called **Linked Lists** and are not natively implemented in Python.

Using Collections

Digital Career Institute

In computer science, collections are objects with properties and methods.

READ	**ADD**	**REMOVE**	**SIZE**

Different types of collections may have different methods,
but reading, adding and removing elements are common.

Adding is commonly referred as **Push**, removing as **Pop**,
reading as **Lookup**, and the size as **Length**.

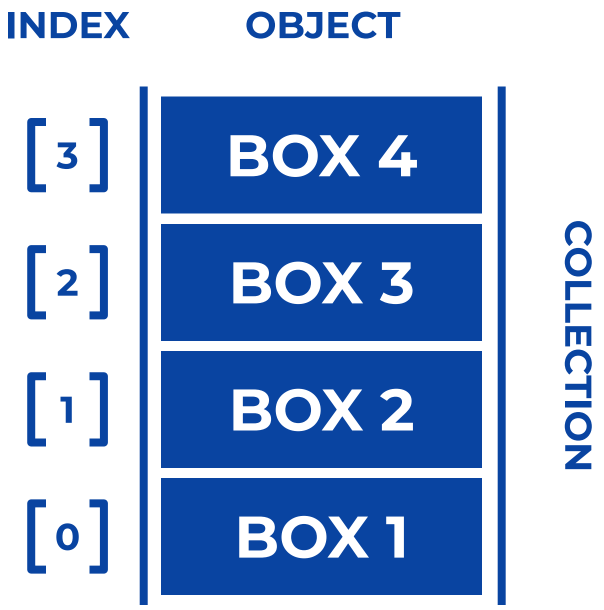
Reading Linear Collections

The objects in a linear collection can be accessed using numeric indexes that identify their position in the sequence.

```
collection[0] = Box 1
```

Indexes start at 0 and increase by 1.

They are not permanently associated to the object. If `Box 1` is removed, the rest of the boxes drop one position and the object associated with the index 0 will be `Box 2`.

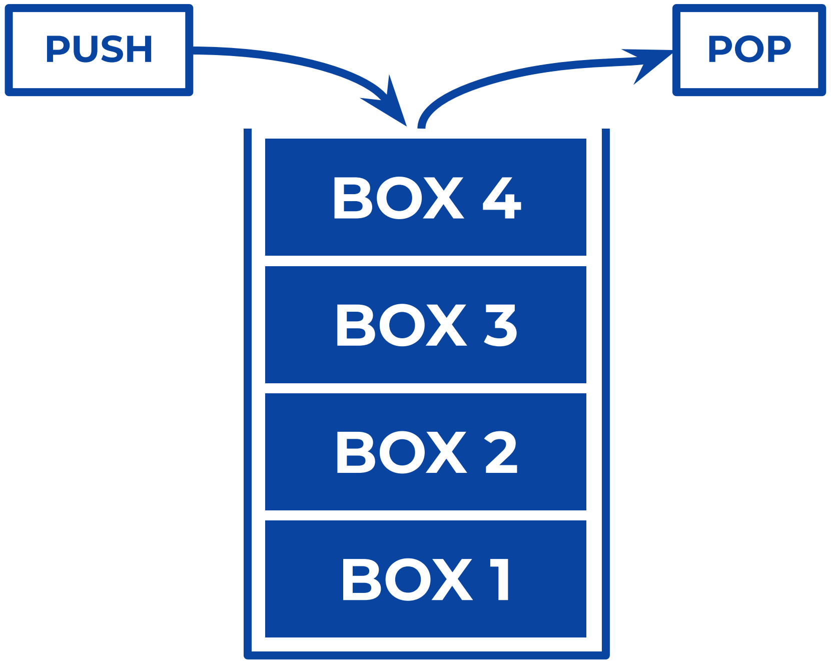


Types of Linear Collections: Stacks

A pile of items is called a **stack** in computer science.

Stacks are linear collections where items can be added at one end and removed from the same end.

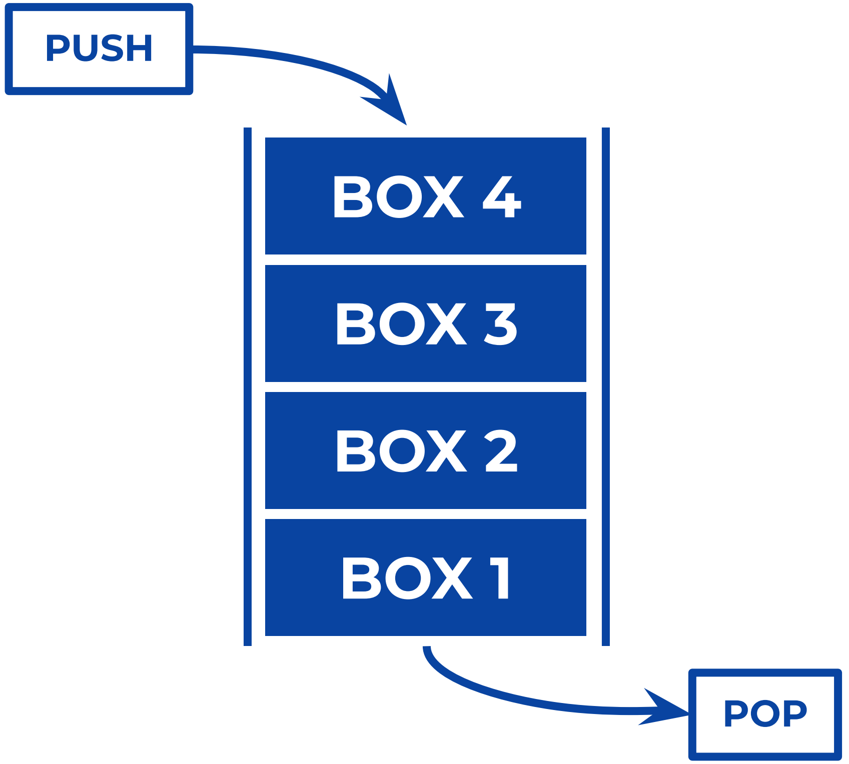
This method of manipulating linear collections is called **LIFO** (Last In, Frist Out).



Types of Linear Collections: Queues

A **queue** is a linear collection that, instead, adds the new items on one end and removes them from the other.

This method of manipulating linear collections is called **FIFO** (First In, Frist Out).

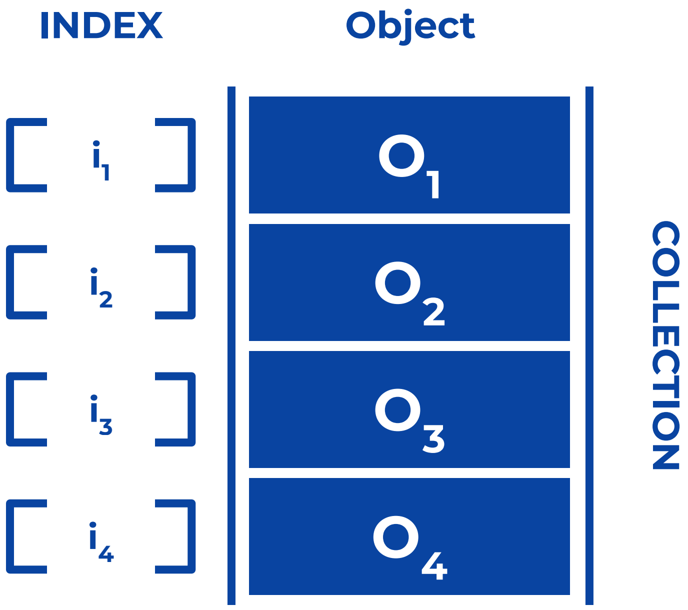


Storing Collections: Associative

Using linear collections implicitly defines an order, and adding or removing objects may change the index associated to each value.

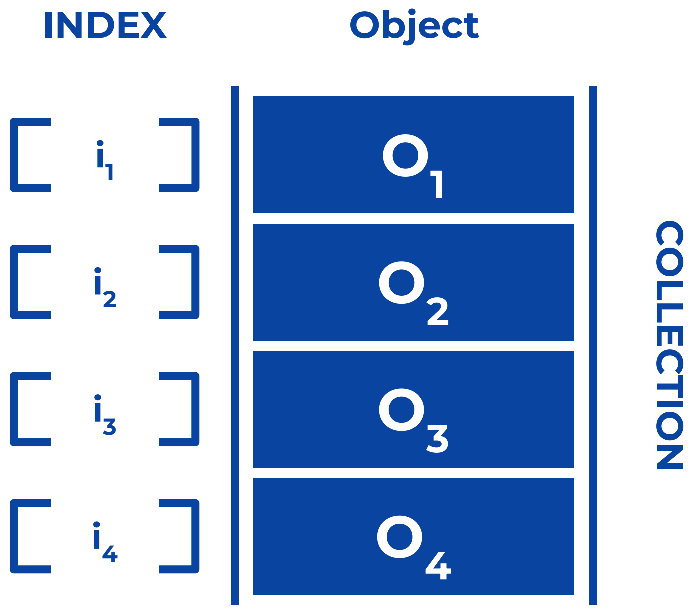
Associative Collections provide a method to assign each index to the same value, even if the rest of the objects are removed.

The index `i`$_2$ will always link to the object `o`$_2$, even if the object `o`$_1$ is removed from the collection.



Storing Collections: Associative

Examples:
- A dictionary.
- The shopping list.
- The ingredients in a recipe.
- A user profile.



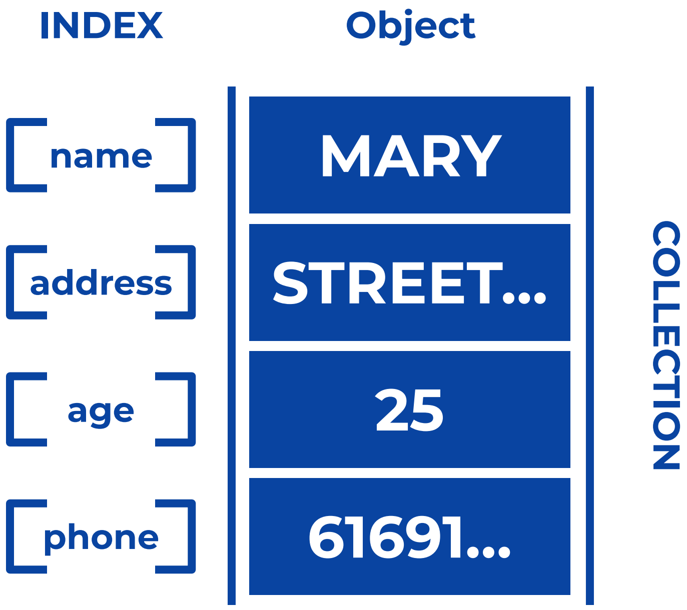
Types of Associative Collections: Arrays

Associative arrays are collections that use a key to index each one of its elements.

This key is usually a string that defines the value associated with it. It is also often called a collection of **key-value** pairs.

Associative arrays often have methods to see if a key exists, to remove a key, access all keys, ...

Example: a user profile.

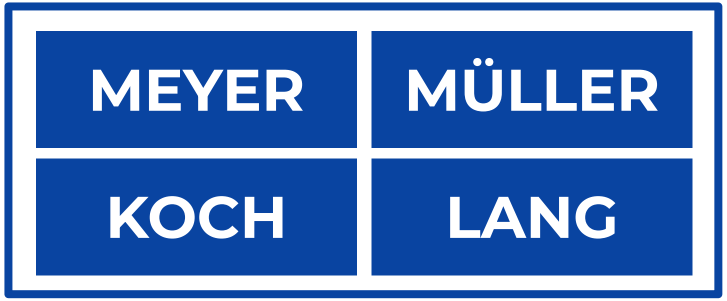


Types of Associative Collections: Sets

A **Set** is an unordered collection where duplicates are not allowed. They replicate the behavior of sets from **Set Theory**.

Sets often have methods to see if a value exists in the set and also provide specific methods to add and remove objects.

Example: german family names.

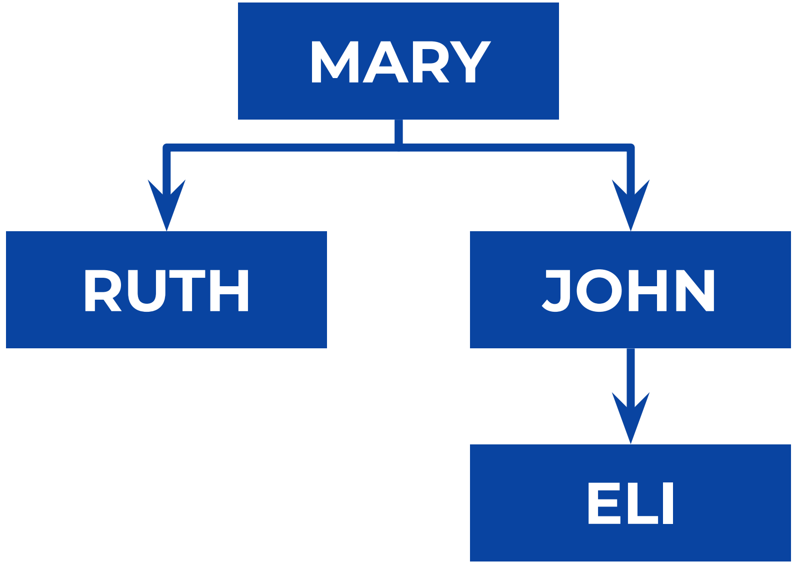


Storing Collections: Graphs

Graphs are collections of node objects, each of which refers to one or more other objects and has its own properties and methods to add, remove and read its elements.

Examples:

- The road network.
- The fiber network.
- The employees in a company.
- The genealogy.



Types of Collections: Summary

- **LINEAR**
  - Linked Lists
  - Stacks
  - Queues
- **ASSOCIATIVE**
  - Arrays
  - Sets
- **GRAPHS**

We learned ...

- What is a collection.
- That there are different types of collections: linear, associative and graphs.
- That linear collections have an order, which can be implicit or explicit (linked lists).
- How to access and manipulate stacks and queues.
- That removing the first object of a linear collection will change the index associated to the rest of the objects.
- That associative collections use other kinds of indexes to access their objects.
- That sets do not allow duplicate values in the collection.

Self Study

- Explore the different types of Python built-in collections:
  - Strings
  - Lists
  - Tuples
  - Sets
  - Dictionaries

Digital Career Institute

Collections in Python

Digital Career Institute

Collections in Python

Digital Career Institute

LINEAR

- LIST
- TUPLE

ASSOCIATIVE

- DICTIONARY
- SET

Lists

Collections in Python

Python Lists

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

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

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

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.

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

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

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

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

Python Lists: Changing Values

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

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

Digital Career Institute

FRIDGE	SENSOR DATA	GOLD MEDALS
	For instance, temperature.	
- Apple	- 15	- M. Mayer
- Carrot	- 15	- D. Hermann
- Iogurt	- 16	- J. Ludwig
- Apple	- 17	- Z.W. Ren
- Iogurt	- 19	- Z.W. Ren
- Beer	- 20	- U. Bogataj
- Steak	- 22	- U. Bogataj
- Cabbage	- 22	- A. Fontana
- Eggplant	- 20	- J.T. Boe
- Orange Juice	- 21	- 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']
```