

Digital Career Institute

Python Course - Collections



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 `()`.

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.

Python Tuples: Accessing Values

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.

Python Tuples: Accessing Values

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.

Python Tuples: Changing Values

The items inside a tuple cannot be changed.

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

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

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

Python Tuples: Use Case Examples

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

Tuples only have methods to search and analyze the values:

- Index
- Count

Sets

Collections in Python

Sets distinctive feature is that they do not allow duplicate values.

They:

- Have an **no order**.
- **Do not store duplicate** values.
- **Do not allow** their objects to be **changed**.
- Allow objects of **different types**.

Defining Sets

They are defined using curly brackets `{}`.

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

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

Defining Sets

Sets can contain values of any type. Repeated values can be added, but they will only be stored once.

Items in the set can be of mixed types.

But the items themselves can not be sets.

A set can also be created using the `copy` method of another set.

Python Sets: Accessing Values

Printing the set will show its values in a **different order** than the one used when the set was defined.

The values in a set cannot be accessed directly using indexing.

Therefore, there is no **index** method. Because the set has no order, it has no method **sort** or **reverse**. And because the set has no repeated values, it does not have the **count** method either.

Python Sets: Adding Values

Adding new values can be done with the method **add**. The new value may show anywhere, as there is no order.

The **update** method can be used to add multiple values at once. It accepts any iterable as an argument.

Adding a value twice throws no error, but only stores the value once.

Python Sets: Removing Values

The method **pop** will remove a random element and will return it. It accepts no argument.

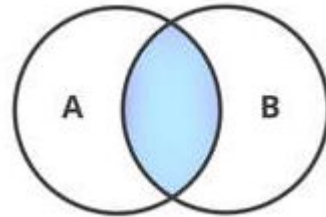
The **discard** method removes the given value and returns nothing.

The **remove** method is like **discard** but it throws an error exception if the value does not exist in the set.

The **clear** method will remove all items in the set.

Python Sets: Set Operations

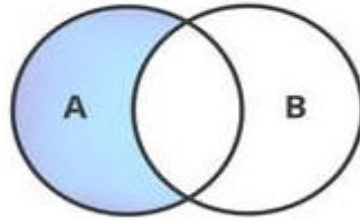
The set type includes methods to perform the standard operations between sets of Set Theory and return new sets.



A and B

The method **intersection** returns a set containing the values present in both sets. The original sets remain unchanged.

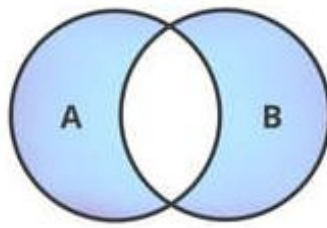
Python Sets: Set Operations



A - B

The method **difference** returns a set containing the values present in A that don't exist in B. The original sets remain unchanged.

Python Sets: Set Operations



A xor B

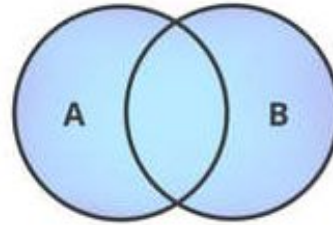
The method **`symmetric_difference`** returns a set containing the values present in A or B that don't exist at the same time in both. The original sets remain unchanged.

Python Sets: Set Operations

The `difference_update`, `intersection_update` and `symmetric_difference_update` methods will calculate the same thing as do the `difference`, `intersection` and `symmetric_difference` methods.

The difference is that the first methods will overwrite the original set (fruits, in this case) instead of returning the result.

Python Sets: Set Operations



A or B

The method **union** returns a set containing the values present in A or B (or both). The original sets remain unchanged.

Python Sets: Set Operations

The previous methods return a set with the result. Sometimes the script only requires to know if some kind of relationships exists between the sets, not the particular values.

The method `isdisjoint` returns `True` if the two sets have no item in common and `False` if they share at least one value.

Python Sets: Set Operations

The method `issubset` returns `True` if the first set is completely contained in the set passed as argument.

The method `issuperset` returns `True` if the first set completely contains the set passed as argument.

Comparing Python Sets

Comparing two sets will return **True** if the following statement is true for the elements in it:

- They are the same

A set is very useful when we need to know if the items in two iterables are the same.

*Using **set()** on both iterables before comparing them will remove duplicates and ignore the order.*

Python Sets: Use Case Examples

CITIES VISITED

- Berlin
- Barcelona
- Stockholm
- Trondheim
- Salzburg
- Brno
- Girona
- Manchester
- Ljubljana
- Tijuana

REGISTERED

- Mary
- John
- Eva
- Susie
- Peter
- Lucy
- Ronnie
- Gerald
- Anna
- Anthony

SPORTS

- Badminton
- Tennis
- Athletics
- Swimming
- Basketball
- Football
- Baseball
- Table-tennis
- Skiing
- Curling

Python Set Methods: Summary

Sets have a long number of methods:

Add

- Add
- Update

Remove

- Pop
- Remove
- Discard
- Clear

Manage

- Copy

Set Operations

- Intersection
- Difference
- ...



THANK YOU

Contact Details
DCI Digital Career Institute gGmbH