Python Sets Cheat Sheet

Learn computer science with Python:

www.k0nze.dev

Sets

Python sets are an unordered data type that resemble mathematical sets. Sets can contain elements of different data types but no duplicates.

Defining a Set

```
>>> s = {True, 42, 0.23, "Hi"}
>>> s
  {True, 42, 0.23, 'Hi'}
>>> l = [False, 23, 0.4, 23, 0.4, "Hello"]
>>> s = set(l)
>>> s
  {False, 'Hello', 0.4, 23}
```

Retrieving Set Elements

Because sets don't have an order, it is impossible to access individual elements through an index or key. However, it is possible to check if an element is in a set with the `in` operator:

```
>>> s = {True, 42, 0.23, "Hi"}
>>> True in s
  True
>>> 42 in s
  True
>>> 'z' in s
  False
```

Adding Set Elements

```
>>> s = {True, 42, 0.23, "Hi"}
>>> s.add('z')
>>> s
  {True, 42, 0.23, "Hi", 'z'}
```

.copy()

Returns a copy of a set.

Syntax

```
set.copy()
```

Parameters

None

Return Value

Copy of the set

Copy vs. Reference

A reference to an object is created with the = operator. When the `.copy()` functions is called on an object the object is duplicated and returned.

Multiple variables reference a single object:

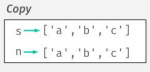
```
>>> s = {'a','b','c','d'}
>>> m = s
>>> m
  {'a','b','c','d'}
>>> s.add('x')
>>> s
  {'a','b','c','d','x'}
>>> m
  {'a','b','c','d','x'}
```

An object copy is created and assigned to a new variable:

```
>>> s = {'a','b','c','d'}
>>> n = s.copy()
>>> n
  {'a','b','c','d'}
>>> s.add('x')
>>> s
  {'a','b','c','d','x'}
>>> n
  {'a','b','c','d'}
```

Reference

Copy



.pop()

Removes an arbitrary element from a set. If the set is empty a `KeyError` is raised.

Syntax

```
set.pop()
```

Parameters

None

Return Value

the element that was removed from the set

```
>>> s = {'a','b'}
>>> s.pop()
  'a'
>>> s
  {'b'}
>>> s.pop()
  'b'
>>> s
  set()
>>> s.pop()
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  KeyError: 'pop from an empty set'
```

.remove()

Removes an element from a set. If the element that should be removed is not present in the set, a KeyError is raised.

Syntax

```
set.remove(element)
```

Parameters

`element:` element that is removed from the set

Return Value

None

```
>>> s = {'a','b','c'}
>>> s.remove('a')
>>> s
  {'b','c'}
>>> s.remove('z')
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  KeyError: 'z'
```

.discard()

Removes an element from a set. If the element that should be removed is not present, no error is raised.

Syntax

```
set.discard(element)
```

Parameters

`element:` element that is removed from the set

Return Value

None

```
>>> s = {'a','b','c'}
>>> s.discard('a')
>>> s
  {'b','c'}
>>> s.discard('z')
>>> s
  {'b','c'}
```

.clear()

Removes all elements from a set.

Syntax

```
set.clear()
```

Parameters

None

Return Value

None

```
>>> s = {'a',b','c'}
>>> s.clear()
>>> s
  set()
```

max() & min()

Returns the maximum/minimum element of a set (iterable).

Syntax

```
max(iterable, key=None)
min(iterable, key=None)
```

Parameters

`iterable:` set (iterable) that is sorted
`key:` sorting key

Return Value

maximum/minimum element

```
>>> s = {'a', 'aa', 'b', 'c'}
>>> max(s)
'c'
>>> max(s, key=len)
'aa'
>>> min(s)
'a'
```

len()

Returns the length of a set or any object that implements `__len__(self)`.

Syntax

```
len(object)
```

Parameters

`object:` object that implements `__len__(self)`

Return Value

length of the object

```
>>> s = {'a','b','c','d'}
>>> len(s)
  4
```

Union

The union between two sets contains all elements from both sets.

$$A \cup B = \{x : x \in A \text{ or } x \in B\}$$

Syntax

```
set.union(iterable...)
set.update(iterable...)
set | set
```

Parameters

`iterable...:` iterables whose elements are added to the union

Return Value

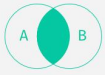
union of the sets/iterables

```
>>> s = {'a','b','c'}
>>> m = {'c','d','e'}
>>> s.union(m)
  {'e', 'd', 'a', 'b', 'c', 'e'}
>>> t = ('e','f','g')
>>> s.union(m, t)
  {'e', 'd', 'a', 'b', 'c', 'e', 'f'}
>>> s.update(m, t)
>>> s
  {'e', 'd', 'a', 'b', 'c', 'e', 'f'}
>>> s = {'a','b','c'}
>>> m = {'c','d','e'}
>>> n = {'e','f','g'}
>>> s | m | n
  {'f', 'g', 'd', 'e', 'a', 'b', 'c'}
```

Intersection

The intersection between two sets contains all elements that are present in both sets.

$$A \cap B = \{x : x \in A \text{ and } x \in B\}$$



Syntax

```
set.intersection(iterable...)
set.intersection_update(iterable...)
```

Parameters

`iterable...:` iterables whose elements are added to the intersection

Return Value

intersection of the sets/iterables (None with _update)

```
>>> s = {'a','b','c'}
>>> m = {'b','c','d'}
>>> s.intersection(m)
  {'b', 'c'}
>>> t = ('c','d','e')
>>> s.intersection(m, t)
  {'c'}
>>> s
  {'a', 'b', 'c'}
>>> s.intersection_update(m, t)
>>> s
  {'c'}
```

Difference

The difference of two sets only contains the elements that are exclusively contained by one set.

$$A \setminus B = \{x : x \in A \text{ and } \neg x \in B\}$$

Syntax

```
set.difference(iterable...)
set.difference_update(iterable...)
```

Parameters

`iterable...:` iterables whose elements the difference is taken from

Return Value

difference of the sets/iterables (None with _update)

```
>>> s = {'a','b','c'}
>>> m = {'a','b','x'}
>>> s.difference(m)
  {'c'}
>>> s.difference_update(m)
>>> s
  {'c'}
>>> s = {'a','b','c','d'}
>>> m = {'a','b','c','x'}
>>> n = {'a','b','c','y'}
>>> s.difference(m, n)
  {'d'}
>>> s.difference_update(m, n)
>>> s
  {'d'}
```

Symmetric Difference

The symmetric difference of two sets includes all elements that are not in both sets.

$$A \triangle B = (A \setminus B) \cup (B \setminus A)$$
$$= (A \cup B) \setminus (A \cap B) = \{x : (x \in A) \oplus (x \in B)\}$$

Syntax

```
set.symmetric_difference(iterable)
set.symmetric_difference_update(iterable)
```

Parameters

`iterable:` iterable whose elements are added to the symmetric difference

Return Value

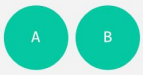
Symmetric difference of the sets (None with _update)

```
>>> s = {'a','b','c','d'}
>>> m = {'b','c','d','e'}
>>> s.symmetric_difference(m)
  {'e', 'a'}
>>> s.symmetric_difference_update(m)
>>> s
  {'e', 'a'}
```

.isdisjoint()

Two sets are disjoint if and only if they have no elements in common.

$$A \cap B = \emptyset$$



Syntax

```
set.isdisjoint(iterable)
```

Parameters

`iterable:` iterables whose elements are checked to determine disjointness

Return Value

boolean if the two sets are disjoint

```
>>> s = {'a','b','c'}
>>> m = {'e','f','g'}
>>> s.isdisjoint(m)
  True
>>> n = {'c','d','e'}
>>> s.disjoint(n)
  False
```

Sub & Superset

All elements of a subset are also present in the superset.

$$A \subseteq B : \forall x (x \in A \implies x \in B)$$

Syntax

```
set.issubset(iterable)
set.issuperset(iterable)
```

Parameters

`iterable:` iterable for which the sub/superset property is checked

Return Value

boolean if the set is a sub/superset

```
>>> s = {'a','b','c'}
>>> m = {'a','b','c'}
>>> n = {'a','b'}
>>> m.issubset(s)
  True
>>> s.issubset(m)
  True
>>> s.issubset(n)
  False
>>> s.issuperset(n)
  True
```

Set of Sets

A set can't contain sets as elements.

```
>>> s = {{1,2},{3,4,5},{6,7},{8}}
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  TypeError: unhashable type: 'set'
```


k0nze builds
k0nze_
discord.k0nze.dev
patreon.com/k0nze

www.k0nze.dev