

A set holds each value once, in no particular order, and answers `x in s` instantly. Examples use `a, b = {1, 2, 3}, {2, 3, 4}`.

CREATE

```
{3, 1, 3} → {1, 3}
duplicates vanish

set() → set()
the only way to write an empty set

set([1, 2, 2, 3]) → {1, 2, 3}
any iterable to a set

sorted(set("hello"))
→ ['e', 'h', 'l', 'o']
sorted() for a predictable order

{n % 3 for n in range(10)} → {0, 1, 2}
set comprehension

frozenset([1, 2]) → frozenset({1, 2})
a set that can't change
```

ADD AND REMOVE

```
s = {1, 2}

s.add(3)
one item; already there? nothing happens

s.update([4, 5])
each item of any iterable

s.discard(9) remove if present, no error

s.remove(5)
remove, but KeyError if missing

s → {1, 2, 3, 4}
after the four lines above

s.remove(9) → raises KeyError
use discard() when it may be missing

s.pop() removes and returns any one item

s.clear() empty it
```

MEMBERSHIP

```
banned = {"spam", "eggs"}
words = ["ham", "spam", "jam", "ham"]

"spam" in banned → True
as fast with 10 million items as with 10

"ham" not in banned → True

[w for w in words if w not in banned]
→ ['ham', 'jam', 'ham']
filter a list against a set

len(set(words)) → 3
how many distinct values

banned.isdisjoint(words) → False
does the list hold any of them?
```

SET ALGEBRA

```
a, b = {1, 2, 3}, {2, 3, 4}
groups = [{1, 2}, {2, 3}, {2, 4}]

a | b → {1, 2, 3, 4}
union: in either

a & b → {2, 3}
intersection: in both

a - b → {1}
difference: in a, not in b

b - a → {4}
order matters for -

a ^ b → {1, 4}
in exactly one of them

a.union([4, 5]) → {1, 2, 3, 4, 5}
methods take any iterable; operators need sets

a.intersection(b, {3, 9}) → {3}
methods take several at once

set.intersection(*groups) → {2}
common to every set in a list
```

UPDATE IN PLACE

```
a, b = {1, 2, 3}, {2, 3, 4}

a |= b add b's items; same as a.update(b)

a &= {2, 3, 9} keep only the shared items

a -= {3} remove all of these

a → {2}
after the three lines above

b ^= {4, 7}
b → {2, 3, 7}
flip: drop the shared, add the new
```

COMPARE

```
{1, 2} <= {1, 2, 3} → True
subset; also issubset()

{1, 2} < {1, 2} → False
proper subset: must be smaller

{1, 2, 3} >= {3} → True
superset; also issuperset()

{1, 2}.issubset([1, 2, 5]) → True
the method takes any iterable

{1, 2} == {2, 1} → True
order never matters

{1, 2}.isdisjoint({3, 4}) → True
nothing in common
```

LOOP AND SUMMARISE

```
a, b = {1, 2, 3}, {2, 3, 4}

for x in sorted(a):
    print(x)
loop in a predictable order

len(a) → 3

min(a), max(a), sum(a) → (1, 3, 6)
the usual built-ins work

sorted(a, reverse=True) → [3, 2, 1]
sorted() always returns a list
```

FROZENSET

```
teams = {frozenset({"ann", "bo"}): "A"}

teams[frozenset({"bo", "ann"})] → 'A'
hashable: a key where order doesn't matter

frozenset({1, 2}) | {3}
→ frozenset({1, 2, 3})
algebra works, the result stays frozen

{frozenset({1, 2}), frozenset({2, 1})}
→ {frozenset({1, 2})}
a set of sets has to use frozenset
```

HANDY PATTERNS

```
xs, ys = [3, 1, 3, 2], [2, 3, 5]

sorted(set(xs)) → [1, 2, 3]
distinct values, in order

list(dict.fromkeys(xs)) → [3, 1, 2]
remove duplicates, keep first-seen order

len(xs) == len(set(xs)) → False
are all the items unique?

set(xs) & set(ys) → {2, 3}
items in both lists

set(ys) - set(xs) → {5}
new in ys

set(xs) | set(ys) → {1, 2, 3, 5}
everything from both, once each

max(set(xs), key=xs.count) → 3
the most common value

{w.lower() for w in ["Hi", "hi", "HI"]}
→ {'hi'}
unique, ignoring case
```

WATCH OUT

```
s = {3, 1}
xs = [3, 1, 3]

type({}) → <class 'dict'>
{} is an empty dict: write set()

{[1, 2]} → raises TypeError
items must be hashable: use the tuple (1, 2)

{"hello"} == set("hello") → False
set("hello") splits into letters; {"hello"} doesn't

s[0] → raises TypeError
no order, no index: sorted(s)[0]

{1, 2} | [3] → raises TypeError
operators need two sets: .union([3]) takes a list

{1, True, 1.0} → {1}
equal values count once, even across types

list(set(xs))
drops duplicates but loses the order: dict.fromkeys(xs)

frozenset({1}).add(2)
→ raises AttributeError
frozen sets can't change

s = s.add(4)
add() returns None, so now s is None
```