

A dict maps unique, hashable keys to values: lookup by key is fast, and keys stay in the order you added them. Examples use `d = {"a": 1, "b": 2}`.

CREATE

```
dict(name="Ada", age=36)
→ {'name': 'Ada', 'age': 36}
keyword style: keys must be valid names
```

```
dict([("a", 1), ("b", 2)])
→ {'a': 1, 'b': 2}
from a list of pairs
```

```
dict(zip("ab", [1, 2]))
→ {'a': 1, 'b': 2}
pair up keys and values
```

```
dict.fromkeys(["x", "y"], 0)
→ {'x': 0, 'y': 0}
same starting value for every key
```

```
type({})
→ <class 'dict'>
{} is an empty dict; an empty set is set()

{(0, 1): "start"}[(0, 1)]
→ 'start'
tuples can be keys, lists can't
```

READ

```
d = {"a": 1, "b": 2}
```

```
d["a"]
→ 1
```

```
d["z"]
→ raises KeyError
missing key raises
```

```
d.get("z")
→ None
missing key gives None
```

```
d.get("z", 0)
→ 0
your default; d is not changed
```

```
"a" in d
→ True
checks keys, not values
```

```
1 in d.values()
→ True
scans every value: slow on big dicts
```

```
len(d)
→ 2
number of keys
```

ADD AND CHANGE

```
d = {"a": 1, "b": 2}
```

```
d["c"] = 3
adds the key, or replaces its value
```

```
d.update({"a": 9, "e": 5})
merge in place: new values win
```

```
d.setdefault("n", 0)
→ 0
returns the value, inserting it if missing
```

```
list(d)
→ ['a', 'b', 'c', 'e', 'n']
updating a key keeps its place
```

MERGE

```
a = {"x": 1, "y": 2}
b = {"y": 20, "z": 3}
```

```
a | b
→ {'x': 1, 'y': 20, 'z': 3}
new dict, right side wins; a |= b does it in place
```

```
b | a
→ {'y': 2, 'z': 3, 'x': 1}
swap sides to let a win
```

```
{**a, **b}
→ {'x': 1, 'y': 20, 'z': 3}
older spelling, same result
```

REMOVE

```
d = {"a": 1, "b": 2, "c": 3}
```

```
d.pop("a")
→ 1
removes the key and returns its value
```

```
d.pop("z", None)
→ None
with a default: no KeyError
```

```
d.popitem()
→ ('c', 3)
the pair added last
```

```
del d["b"]
KeyError if missing
```

LOOP

```
d = {"a": 1, "b": 2}
```

```
for key in d:
    print(key)
a plain loop gives the keys
```

```
for key, value in d.items():
    print(key, value)
keys and values together: the usual loop
```

```
list(d)
→ ['a', 'b']
the keys
```

```
list(d.values())
→ [1, 2]
```

```
list(d.items())
→ [('a', 1), ('b', 2)]
```

```
next(iter(d))
→ 'a'
the first key
```

```
max(d, key=d.get)
→ 'b'
the key with the biggest value
```

SORT

```
d = {"pear": 3, "fig": 1, "kiwi": 2}
```

```
sorted(d)
→ ['fig', 'kiwi', 'pear']
the keys, sorted
```

```
dict(sorted(d.items()))
→ {'fig': 1, 'kiwi': 2, 'pear': 3}
a new dict in key order
```

```
sorted(d.items(), key=lambda kv: kv[1])
→ [('fig', 1), ('kiwi', 2), ('pear', 3)]
pairs ordered by value
```

```
sorted(d, key=d.get, reverse=True)[:2]
→ ['pear', 'kiwi']
top 2 keys by value
```

COMPREHENSIONS

```
d = {"a": 1, "b": 2, "c": 0}
```

```
{k: v * 10 for k, v in d.items()}
→ {'a': 10, 'b': 20, 'c': 0}
change every value
```

```
{k: v for k, v in d.items() if v}
→ {'a': 1, 'b': 2}
keep only some pairs
```

```
{v: k for k, v in d.items()}
→ {1: 'a', 2: 'b', 0: 'c'}
swap; a repeated value keeps the last key
```

```
{k: d[k] for k in ("a", "c") if k in d}
→ {'a': 1, 'c': 0}
pick out a few keys
```

COUNT AND GROUP

```
from collections import Counter, defaultdict
words = ["fig", "kiwi", "fig", "pear"]
```

```
Counter(words)
→ Counter({'fig': 2, 'kiwi': 1, 'pear': 1})
the shortcut for counting
```

```
Counter(words).most_common(1)
→ [('fig', 2)]
```

```
groups = defaultdict(list)
for w in words:
    groups[len(w)].append(w)
a missing key starts as []
```

```
dict(groups)
→ {3: ['fig', 'fig'], 4: ['kiwi', 'pear']}
words grouped by length
```

NESTED

```
user = {"addr": {"city": "Oslo"}, "langs": ["py"]}
```

```
user["addr"]["city"]
→ 'Oslo'
```

```
user.get("job", {}).get("title")
→ None
safe when a level may be missing
```

```
user["langs"].append("sql")
changes the inner list in place
```

VIEWS AND COPIES

```
d = {"a": 1, "b": 2}
```

```
keys = d.keys()
d["c"] = 3
keys
→ dict_keys(['a', 'b', 'c'])
views are live: they follow d
```

```
c = d.copy()
shallow: inner lists are still shared
```

```
import copy
c = copy.deepcopy(d)
copies everything inside too
```

WATCH OUT

```
d = {"a": 1, "b": 2}
```

```
for k in d:
    del d[k]
RuntimeError: loop over list(d) to add or remove keys
```

```
d[["x", "y"]] = 1
TypeError: a list can't be a key; use a tuple
```

```
d.keys()[0]
TypeError: views can't be indexed; use list(d)[0]
```

```
{1: "a", 1.0: "b", True: "c"}
→ {1: 'c'}
1, 1.0 and True are the same key
```

```
g = dict.fromkeys("ab", [])
g["a"].append(1)
g
→ {'a': [1], 'b': [1]}
one shared list: use {k: [] for k in "ab"}
```

```
d.get("tags", []).append("x")
the default is never stored: use setdefault
```

```
e = d
e["a"] = 0
e is d, not a copy: d changed too
```