06 – Sets & dictionaries

Bálint Aradi

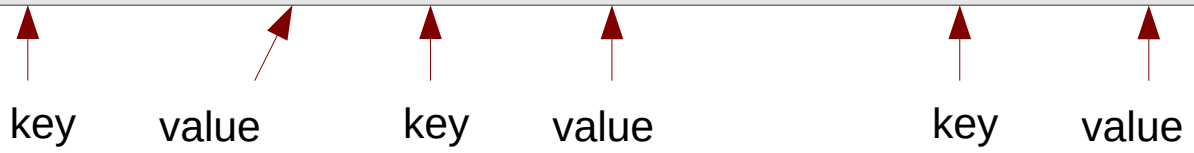
Scientific Programming in Python (2025)

https://atticlectures.net/scipro/python-2025/

Dictionaries

- Store items of arbitrary type
- Items identified by their unique key, not by their position
- Key must be of immutable data type
- Dictionary is delimited by **{** and **}**

```
dd = {"test1": 1, "test2": "Hello", 12: [1, 2]}
dd
{'test1': 1, 12: [1, 2], 'test2': 'Hello'}
```



- Elements can be accessed as in lists, but by using their key

```
dd["test1"]
1
dd[12]
[1, 2]
```

Dictionaries

- Dictionaries are mutable
- If a key is used, which is already present, the item is overwritten

```
dd["test1"] = 3+4j
dd
{'test1': (3+4j), 12: [1, 2], 'test2': 'Hello'}
```

- If a key is used, which is not present yet, a new item is created

```
dd[(-1,)] = 12
dd
{'test1': (3+4j), 12: [1, 2], 'test2': 'Hello', (-1,): 12}
```

- Elements can be deleted by the **del** statement

```
del dd["test2"]
dd
{'test1': (3+4j), 12: [1, 2], (-1,): 12}
```

Dictionaries

- The **in** operator can be used to check the presence of a key

```
'test1' in dd
True
"missing" in dd
False
```

- Trying to access a non-existing key leads to an error

```
dd["missing"]
… KeyError: 'missing
```

- The **get()** method can be used to obtain an item or a default value if the key is not found

```
value = dd.get("missing", -1)
value
-1
```

```
if "missing" in dd:
    value = dd["missing"]
else:
    value = -1
```

Dictionaries as iterators

- Dictionaries return their keys one by one:

```
dd = {12: [1, 2], 'test1': 3.2, (-1,): True}
for key in dd:
    print(f"key: {key}")
```

```
key: 12
key: (-1,)
key: test1
```

- An iterator over dictionary values can be obtained by the **values()** method

```
for val in dd.values():
    print(f"value: {val}")
```

```
value: [1, 2]
value: True
value: 3.2
```

- An iterator over key, value tuples can be obtained by the **items()** method:

```
for key, val in dd.items():
    print(f"{key}: {val}")
```

```
12: [1, 2]
(-1,): True
test1: 3.2
```

Creating dictionaries

- From a dict-literal

```
dd = {3.2: 'hello', 'a': 1}
```

```
{3.2: 'hello', 'a': 1}
```

- From an iterable containing (key, value) tuples

```
dict([('a', 1), (3.2, 'hello')])
```

```
{3.2: 'hello', 'a': 1}
```

- From a dictionary comprehension

filtering is optional

```
{keyexpr: valuexpr for itervar in iterator if condition}
```

```
nums = [1, 3, 2, 9, 8, 3]
oddsquares = {num: num**2 for num in nums if num % 2 == 1}
```

```
{1: 1, 3: 9, 9: 81}
```

Sets

- Sets contain only keys (like dictionaries), but no values
- Sets are mutable
- All members must be of inmutable type
- Every key (element) is unique and occurs only once
- Elements can be added by the **add()** method

```
ss = {"test", 12, -3.6, (1,2)}
ss
{(1, 2), 12, -3.6, 'test'}
```

```
ss.add(True)
ss
{(1, 2), True, 12, -3.6, 'test'}
```

- Adding an already existing element to the set leaves it unchanged:

```
ss.add("test")
ss
{(1, 2), True, 12, -3.6, 'test'}
```

Sets

- Elements can removed by the **remove()** method

```
ss.remove(-3.6)
ss
{(1, 2), True, 12, 'test'}
```

- The **in** operator can be used to check the presence of an element

```
ss
{(1, 2), True, 12, 'test'}
12 in ss
True
13 in ss
False
```

Sets as iterators

- Sets return their elements one by one, but the order is undetermined:

```
ss = {True, 12, 'test', (1, 2)}
for item in ss:
    print('Item:', item)
```

```
Item: (1, 2)
Item: True
Item: 12
Item: test
```

Creating sets

- From a set-literal

```
ss = {1, 9, (3, 4), False, 8.2}
```

```
{1, 9, (3, 4), False, 8.2}
```

- From an iterable containing (key, value) tuples

```
set([1, 9, (3, 4), False, 8.2])
```

```
{1, 9, (3, 4), False, 8.2}
```

- From a set-comprehension

filtering is optional

```
{expr for itervar in iterator if condition}
```

```
nums = [1, 3, 2, 9, 8, 3]
oddsquares = {num**2 for num in nums if num % 2 == 1}
```

```
{1, 9, 81}
```

Containers – overview

Lists

- Ordered
- Elements indexed by sequential integer (position)
- Index of a given element might change when other elements are inserted/deleted
- Fast *O(1)* access by index
- Slow O(N) access by value

Dictionaries

- Unordered (ordered for Python > 3.7)
- Elements indexed by key (arbitrary inmutable object)
- Index of given element remains unchanged when other elements are inserted/deleted
- Fast O(1) by key
- Slow O(N) access by value

Sets

- Unordered
- Elements are unique
- Fast O(1) access for checking element presence

Containers – access times

Note: choice of the container type might seriously affect performance

```
import random
MAX_NUM = 10000000
```

```
random_list = [random.randint(0, MAX_NUM)
               for _ in range(MAX_NUM)]
random_set = set(random_list)
```

```
%%timeit
(MAX_NUM + 1 in random_list)
```

compare execution times!

```
%%timeit
(MAX_NUM + 1) in random_set
```

Comparing containers

- Equality of containers can be checked with **==** and **!=** operators
- Two containers are equal, if all elements and their keys/indices are equal

```
{'key1': 1, 'key2': 2} == {'key2': 2, 'key1': 1}   True
{'key1': 9, 'key2': 2} == {'key2': 2, 'key1': 1}   False
```

- Ordered (sequence) types (lists, tuples, but not dicts) can also be compared by **>**, **>=**, **<**, **<=**
- The comparison is done component-wise
- The first non-matching component determines the relation

```
(1, 2, 3) > (1, 2, 4)          False
("hello", 6) > ("ahoi", 9)     True
```

- The same ordering rules are applied in internal routines, like sorting:

```
ll = [("hello", 6), ("ahoi", 9)]
ll.sort()
ll
```

```
[('ahoi', 9), ('hello', 6)]
```