

Programmation Fonctionnelle : des concepts aux applications web (NFP119)

Dictionnaires*

Tristan Crolard

Department of Computer Science
CEDRIC lab / SYS team

`tristan.crolard@cnam.fr`

`cedric.cnam.fr/sys/crolard`

*. Ces supports sont adaptés de *Python for Computational Science* (2024)

Dictionaries

- ▶ Python provides another data type: the dictionary.
(dictionaries are also called “associative arrays” and “mappings”).
- ▶ Dictionaries are **mutable** sets of key-value pairs
(the insertion order is preserved).
- ▶ An empty dictionary can be created using curly braces:

```
>>> d = {}
```

- ▶ Key-value pairs can be added like this:

```
>>> d["today"] = "22_deg_C" # "today" is the key  
                           # "22_deg_C" is the value
```

```
>>> d["yesterday"] = "19_deg_C"
```

```
>>> d
```

```
{'today': '22 deg C', 'yesterday': '19 deg C'}
```

- ▶ We can retrieve a value by using its key as the index:

```
>>> print(d["today"])
```

```
22 deg C
```

- ▶ `d.keys()` returns the sequence of keys:

```
>>> list(d.keys())
```

```
['today', 'yesterday']
```

- ▶ `d.values()` returns the sequence of values:

```
>>> list(d.values())
```

```
['22 deg C', '19 deg C']
```

- ▶ Check if some key is in the dictionary:

```
>>> 'today' in d.keys()
```

True

Equivalent to:

```
>>> 'today' in d
```

True

Dictionaries – example 1

```
>>> order = {} # create empty dictionary
>>> # add orders as they come in
>>> order["Peter"] = "Sparkling_water"
>>> order["Paul"] = "Half_pint_of_beer"
>>> order["Mary"] = "Gin_tonic"
>>> # deliver order at bar
>>> for person in order.keys():
    print(person, "requests", order[person])
```

```
Peter requests Sparkling water
Paul requests Half pint of beer
Mary requests Gin tonic
```

Dictionaries – example 2

```
>>> # keys are names of people
```

```
>>> # values are the office room numbers
```

```
>>> offices = {"Andy": 1031, "Barbara": 1027, "Charles": 1033}
```

```
>>> for person in offices:  
    print(person, "works in", offices[person])
```

Andy works in 1031

Barbara works in 1027

Charles works in 1033

Without dictionary, we would need a list of pairs (less efficient):

```
>>> offices = [("Andy",1031), ("Barbara",1027), ("Charles",1033)]  
>>> for (person, room) in offices:  
    print(person, "works_in_room", room)
```

Andy works in room 1031

Barbara works in room 1027

Charles works in room 1033

Merging dictionaries

You can merge two dictionaries using operator `|`, for instance:

```
>>> {'Andy': 1031, 'Barbara': 1027} | {'Charles': 1033}
```

```
{'Andy': 1031, 'Barbara': 1027, 'Charles': 1033}
```

You can check if some key already exists in a dictionary using operator `in`.
If some key already exists in the first dictionary, its value is **updated**:

```
>>> {'Andy': 1031, 'Barbara': 1027} | {'Barbara': 1067}
```

```
{'Andy': 1031, 'Barbara': 1067}
```

Dictionary comprehension

In addition to list comprehension, dictionary comprehension is also available:

```
>>> {x: x**2 for x in range(5)}
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
>>> {word: len(word) for word in ["dog", "bird", "mouse"]}
```

```
{'dog': 3, 'bird': 4, 'mouse': 5}
```

Recall this list of pairs:

```
>>> offices
```

```
[('Andy', 1031), ('Barbara', 1027), ('Charles', 1033)]
```

This list can be converted into a dictionary simply as follows:

```
>>> {person: room for (person, room) in offices}
```

```
{'Andy': 1031, 'Barbara': 1027, 'Charles': 1033}
```

Converting between dictionary and lists of pairs

```
>>> def dict_to_list[K, V](d: dict[K, V]) -> list[tuple[K, V]]:  
    return [(k, d[k]) for k in d]
```

```
>>> dict_to_list({'Andy': 31, 'Barbara': 27, 'Charles': 33})  
[('Andy', 31), ('Barbara', 27), ('Charles', 33)]
```

```
>>> def list_to_dict[K, V](l: list[tuple[K, V]]) -> dict[K, V]:  
    return {k: v for (k, v) in l}
```

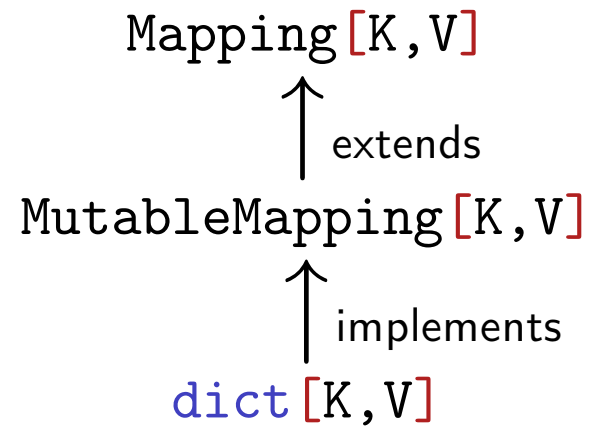
```
>>> list_to_dict([('Andy', 31), ('Barbara', 27), ('Charles', 33)])  
{'Andy': 31, 'Barbara': 27, 'Charles': 33}
```

Mapping types

Abstract Base Class:

Abstract Base Class:

Concrete Class:



Dictionaries – summary

- ▶ The dictionary key must be **immutable** objects, which includes:
 - numbers
 - strings
 - tuples
- ▶ dictionaries are very fast in retrieving values (when given the key)
- ▶ more convenient and more efficient than lists of pairs
- ▶ useful if you have a data set that needs to be indexed by strings or tuples (or other immutable objects)