浙江理工大学
ZHEJIANG SCI-TECH UNIVERSITY

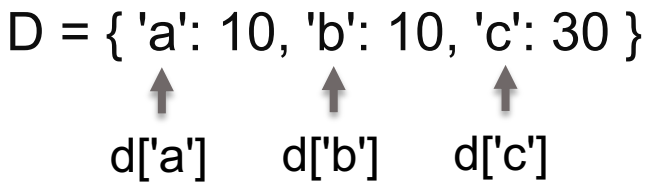
4. Dictionaries, Sets

《Python programming》 / Lecturer : Zhiyi Luo (罗志一)

School of Computer Science and Technology
计算机科学与技术学院

Dictionaries

- Unordered collections of unique values stored in (Key-Value) pairs.



- Unordered: Not reliable, relevant to the item insert order.
- Unique: Each value has a key; the keys in Dictionaries should be unique.
- Mutable: The dictionaries are collections that are changeable, which implies that we can add or remove items after the creation.

Creating a dictionary

- Using curly brackets **{}**
  - The dictionaries are create by enclosing the comma-separated **Key:Value** pairs inside the **{}** curly brackets. The colon **:** is used to separate the key and value in a pair.
- Using **dict()** constructor
  - Create a dictionary by passing the comma-separated **Key:Value** pairs inside the dict().

Creating a dictionary

- Using curly brackets {}

```
>>> person = {"name": "Jessa", "country": "USA", "telephone": 1178}
>>> print(person)
{'name': 'Jessa', 'country': 'USA', 'telephone': 1178}
```

Creating a dictionary

- Using **dict()** constructor

```
>>> person = dict({"name": "Jessa", "country": "USA", "telephone": 1178})
>>> print(person)
{'name': 'Jessa', 'country': 'USA', 'telephone': 1178}
>>> # create a dictionary from sequence having each item as a pair
>>> person = dict([("name", "Mark"), ("country", "USA"), ("telephone", 1178)])
>>> print(person)
{'name': 'Mark', 'country': 'USA', 'telephone': 1178}
>>> # create dictionary with mixed keys keys
>>> # first key is string and second is an integer
>>> sample_dict = {"name": "Jessa", 10: "Mobile"}
>>> print(sample_dict)
{'name': 'Jessa', 10: 'Mobile’}
```

Convert by using dict()

- Using **dict()** to convert two-value sequences into a dictionary. The first item in each sequence is used as the key and the second as the value.

- Remember that the order of keys in a dictionary is arbitrary, and might differ depending on how you add items.

A list of two-item tuples:

```
>>> lot = [ ('a', 'b'), ('c', 'd'), ('e', 'f') ]
>>> dict(lot)
{'c': 'd', 'a': 'b', 'e': 'f'}
```

A list of two-character strings:

```
>>> los = [ 'ab', 'cd', 'ef' ]
>>> dict(los)
{'c': 'd', 'a': 'b', 'e': 'f'}
```

Dictionaries: Hashing

- Quite similar values often have very different hashes.
- Hashes look crazy, but the same value always returns the same hash.
- Keys and Indexes
  - To build an index, Python uses the bottom n bits of the hash.

- A dictionary is really a list.
- Keys are hashed to produce indexes.

Empty Dictionary

- When we create a dictionary without any elements inside the curly brackets then it will be an empty dictionary.

```
>>> emptydict = {}
>>> print(type(emptydict))
<class 'dict'>
```

Add or Change an Item by [key]

- Adding an item to a dictionary is easy. Just refer to the item by its key and assign a value. If the key was already present in the dictionary, the existing value is replaced by the new one.

```
>>> pythons = {
    'Chapman': 'Graham',
    'Cleese': 'John',
    'Idle': 'Eric',
    }
>>> pythons
{'Cleese': 'John', 'Chapman': 'Graham', 'Idle': 'Eric'}
```

```
>>> pythons['Cleese'] = 'Gerry'
>>> pythons
{'Cleese': 'Gerry', 'Chapman': 'Graham', 'Idle': 'Eric’, }

>>> some_pythons = {
...     'Graham': 'Chapman',
...     'Eric': 'Idle',
...     'Terry': 'Gilliam',
...     'Terry': 'Jones',
...     }
>>> some_pythons
{'Terry': 'Jones', 'Eric': 'Idle', 'Graham': 'Chapman'}
```

Combine Dictionaries with update()

```
>>> first = {'a': 1, 'b': 2}
>>> second = {'b': 'platypus'}
>>> first.update(second)
>>> first
{'b': 'platypus', 'a': 1}
```

Delete an Item by Key with del

```
>>> del pythons['Marx']
>>> pythons
{'Cleese': 'John', 'Howard': 'Moe', 'Gilliam': 'Terry',
'Palin': 'Michael', 'Chapman': 'Graham', 'Idle': 'Eric',
'Jones': 'Terry'}
>>> del pythons['Howard']
>>> pythons
{'Cleese': 'John', 'Gilliam': 'Terry', 'Palin': 'Michael',
'Chapman': 'Graham', 'Idle': 'Eric', 'Jones': 'Terry'}
```

Delete All Items by Using clear()

- To delete all keys and values from a dictionary, use clear() or just reassign an empty dictionary ({}) to the name:

```
>>> pythons.clear()
>>> pythons
{}
>>> pythons = {}
>>> pythons
{}
```

Test for a Key by Using in

```
>>> pythons = {'Chapman': 'Graham', 'Cleese': 'John',
'Jones': 'Terry', 'Palin': 'Michael'}
```

Now let’s see who’s in there:

```
>>> 'Chapman' in pythons
True
>>> 'Gilliam' in pythons
False
```

Get an Item by [key]

- You specify the dictionary and key to get the corresponding value:

```
>>> pythons['Cleese']
'John'
If the key is not present in the dictionary, you'll get an
exception:
>>> pythons['Marx']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Marx'
```

- There are two good ways to avoid this. The first is to test for the key at the outset by using in, as you saw in the previous section:

```
>>> 'Marx' in pythons
False
```

- The second is to use the special dictionary get() function. You provide the dictionary, key, and an optional value.

```
>>> pythons.get('Cleese')
'John'
>>> pythons.get('Marx', 'Not a Python')
'Not a Python'
>>> pythons.get('Marx')
>>>
```

Get All Keys by Using keys()

```
>>> signals = {'green': 'go', 'yellow': 'go faster', 'red': 'smile for the camera'}
>>> signals.keys()
dict_keys(['green', 'red', 'yellow’])
>>> list( signals.keys() )
['green', 'red', 'yellow’]
```

Get All Values by Using values()

```
>>> list( signals.values() )
['go', 'smile for the camera', 'go faster']
>>> list( signals.items() )
[('green', 'go'), ('red', 'smile for the camera'), ('yellow', 'go faster')]
```

Each key and value is returned as a tuple, such as ('green', 'go').

Assign with =, Copy with copy()

```
>>> signals = {'green': 'go', 'yellow': 'go faster', 'red': 'smile for the camera'}
>>> save_signals = signals
>>> signals['blue'] = 'confuse everyone'
>>> save_signals
{'blue': 'confuse everyone', 'green': 'go‘, 'red': 'smile for the camera', 'yellow': 'go faster'}
```

- To actually copy keys and values from a dictionary to another dictionary and avoid this, you can use copy():

```
>>> signals = {'green': 'go', 'yellow': 'go faster', 'red': 'smile for the camera'}
>>> original_signals = signals.copy()
>>> signals['blue'] = 'confuse everyone'
>>> signals
{'blue': 'confuse everyone', 'green': 'go',
'red': 'smile for the camera', 'yellow': 'go faster'}
>>> original_signals
{'green': 'go', 'red': 'smile for the camera', 'yellow': 'go faster'}
```

Sets

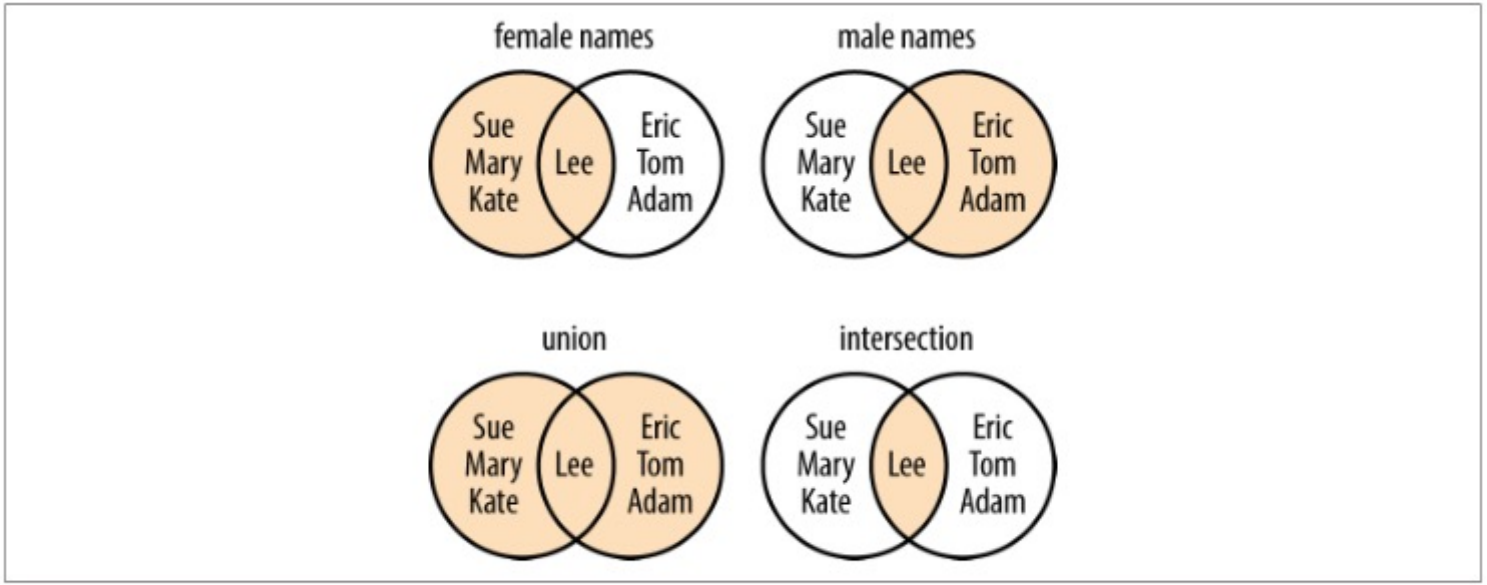


Figure 3-1. Common things to do with sets

Set

- In Python, a Set is an **unordered** collection of data items that are unique. In other words, Python set is a collection of elements (or objects) that contains no duplicate elements.
- So you cannot access elements by their index or perform insert operation using an index number.
- We will learn Set data structure in general, different ways of creating them, and adding, updating, and removing the Set items. We will also learn different set operations.

Create with set()

```
>>> empty_set = set()
>>> empty_set
set()
>>> even_numbers = {0, 2, 4, 6, 8}
>>> even_numbers
{0, 8, 2, 4, 6}
>>> odd_numbers = {1, 3, 5, 7, 9}
>>> odd_numbers
{9, 3, 1, 5, 7}
```

Convert from Other Data Types with set()

- You can create a set from a list, string, tuple, or dictionary, discarding any duplicate values.

```
>>> set( 'letters' )
{'l', 'e', 't', 'r', ‘s’}
```

- Now, let’s make a set from a list:

```
>>> set( ['Dasher', 'Dancer', 'Prancer', 'Mason-Dixon'] )
{'Dancer', 'Dasher', 'Prancer', 'Mason-Dixon'}
```

- This time, a set from a tuple:

```
>>> set( ('Ummagumma', 'Echoes', 'Atom Heart Mother') )
{'Ummagumma', 'Atom Heart Mother', 'Echoes’}
```

- When you give set() a dictionary, it uses only the keys:

```
>>> set( {'apple': 'red', 'orange': 'orange', 'cherry': 'red'} )
{'apple', 'cherry', 'orange'}
```

Test for Value by Using in

```
>>> drinks = {
...     'martini': {'vodka', 'vermouth'},
...     'black russian': {'vodka', 'kahlua'},
...     'white russian': {'cream', 'kahlua', 'vodka'},
...     'manhattan': {'rye', 'vermouth', 'bitters'},
...     'screwdriver': {'orange juice', 'vodka'}
...     }
```

```
>>> for name, contents in drinks.items():
...     if 'vodka' in contents:
...         print(name)
...
screwdriver
martini
black russian
white Russian

>>> for name, contents in drinks.items():
...     if 'vodka' in contents and not ('vermouth' in contents or
...         'cream' in contents):
...         print(name)
...
screwdriver
black russian
```

Combinations and Operators

- The result of the & operator is a set, which contains all the items that appear in both lists that you compare. If neither of those ingredients were in contents, the & returns an empty set, which is considered False.

```
>>> for name, contents in drinks.items():
...     if contents & {'vermouth', 'orange juice'}:
...         print(name)
...
screwdriver
martini
manhattan
```

```
>>> for name, contents in drinks.items():
...     if 'vodka' in contents and not contents & {'vermouth',
'cream'}:
...         print(name)
...
screwdriver
black Russian
>>> bruss = drinks['black russian']
>>> wruss = drinks['white russian']
```

- The following are examples of all the set operators.

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

- You get the intersection (members common to both sets) with the special punctuation symbol & or the set intersection() function

```
>>> a & b
{2}
>>> a.intersection(b)
{2}
>>> bruss & wruss
{'kahlua', 'vodka'}
```

- In this example, you get the union (members of either set) by using | or the set union() function:

```
>>> a | b
{1, 2, 3}
>>> a.union(b)
{1, 2, 3}
```

- And here’s the alcoholic version:

```
>>> bruss | wruss
{'cream', 'kahlua', 'vodka'}
```

- The difference (members of the first set but not the second) is obtained by using the character - or difference():

```
>>> a - b
{1}
>>> a.difference(b)
{1}
>>> bruss - wruss
set()
>>> wruss - bruss
{'cream'}
```

- The exclusive or (items in one set or the other, but not both) uses ^ or symmetric_difference():

```
>>> a ^ b
{1, 3}
>>> a.symmetric_difference(b)
{1, 3}
```

- Y ou can check whether one set is a subset of another by using <= or issubset():

```
>>> a <= b
False
>>> a.issubset(b)
False
```

- And finally, you can find a proper superset (the first set has all members of the second, and more) by using >:

```
>>> a > b
False
>>> a > a
False
```

Compare Data Structures

- You make a list by using square brackets ([]), a tuple by using commas, and a dictionary by using curly brackets ({}).

```
>>> marx_list = ['Groucho', 'Chico', 'Harpo']
>>> marx_tuple = 'Groucho', 'Chico', 'Harpo'
>>> marx_dict = {'Groucho': 'banjo', 'Chico': 'piano', 'Harpo': 'harp'}
>>> marx_list[2]
'Harpo'
>>> marx_tuple[2]
'Harpo'
>>> marx_dict['Harpo']
'harp'
```

Make Bigger Data Structures

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> pythons = ['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin']
>>> stooges = ['Moe', 'Curly', 'Larry’]
```

- We can make a tuple that contains each list as an element:

```
>>> tuple_of_lists = marxes, pythons, stooges
>>> tuple_of_lists
(['Groucho', 'Chico', 'Harpo'],
['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin'],
['Moe', 'Curly', 'Larry'])
```

- We can make a list that contains the three lists:

```
>>> list_of_lists = [marxes, pythons, stooges]
>>> list_of_lists
[['Groucho', 'Chico', 'Harpo'],
['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin'],
['Moe', 'Curly', 'Larry']]
```

- Finally, let's create a dictionary of lists.

```
>>> dict_of_lists = {'Marxes': marxes, 'Pythons': pythons, 'Stooges': stooges}
>> dict_of_lists
{'Stooges': ['Moe', 'Curly', 'Larry'],
'Marxes': ['Groucho', 'Chico', 'Harpo'],
'Pythons': ['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin']}
```

Set

S = { 20, ‘Jessa’, 35.75, [20, 30, 40] }

- Unordered: Set doesn’t maintain the order of the data insertion. The items will be in different order each time when we access the Set object. There will not be any index value assigned to each item in the set.
- Immutable: Set are immutable and we can’t modify items. Set items must be immutable. We cannot change the set items, i.e., We cannot modify the items’ value. But we can add or remove items to the Set. A set itself may be modified, but the elements contained in the set must be of an immutable type.

Set

S = { 20, ‘Jessa’, 35.75 }

- Heterogeneous: Set can contains data of all types (immutable).
- Unique: Set doesn’t allows duplicates items