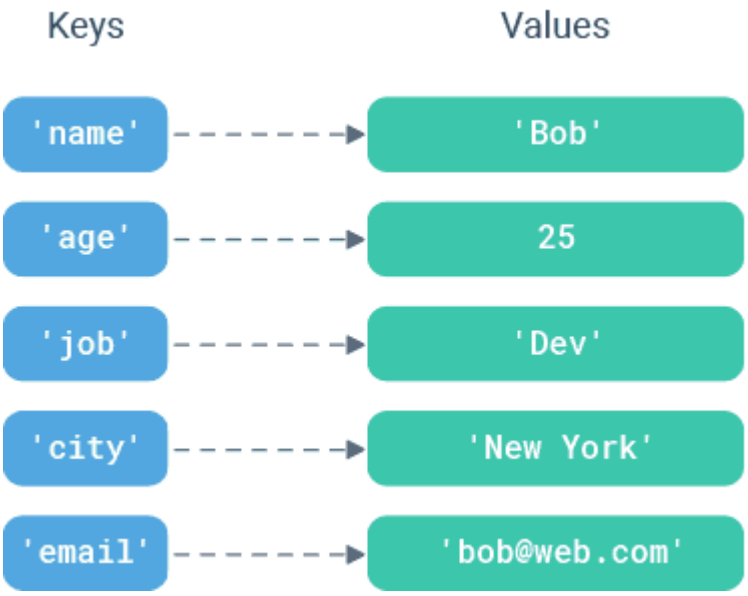
Python Dictionary

Dictionaries are Python's implementation of a data structure, generally known as associative arrays, hashes, or hashmaps.

You can think of a dictionary as a mapping between a set of indexes (known as keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key:value pair or sometimes an item.

As an example, we'll build a dictionary that stores employee record.



Create a Dictionary

You can create a dictionary by placing a comma-separated list of `key:value` pairs in curly braces `{}`. Each key is separated from its associated value by a colon `:`

```
# Create a dictionary to store employee record
D = {'name': 'Bob',
    'age': 25,
    'job': 'Dev',
    'city': 'New York',
    'email': 'bob@web.com'}
```

The dict() Constructor

You can convert two-value sequences into a dictionary with Python's dict() constructor. The first item in each sequence is used as the key and the second as the value.

```
# Create a dictionary with a list of two-item tuples
L = [('name', 'Bob'),
    ('age', 25),
    ('job', 'Dev')]

D = dict(L)
print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev'}

# Create a dictionary with a tuple of two-item lists
T = (['name', 'Bob'],
    ['age', 25],
    ['job', 'Dev'])

D = dict(T)
print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

When the keys are simple strings, it is sometimes easier to specify key:value pairs using keyword arguments.

```
D = dict(name = 'Bob',
        age = 25,
        job = 'Dev')

print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

Other Ways to Create Dictionaries

There are lots of other ways to create a dictionary.

You can use `dict()` function along with the zip() function, to combine separate lists of keys and values obtained dynamically at runtime.

```
# Create a dictionary with list of zipped keys/values
keys = ['name', 'age', 'job']
values = ['Bob', 25, 'Dev']

D = dict(zip(keys, values))

print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev'}
```

You'll often want to create a dictionary with default values for each key. The fromkeys() method offers a way to do this.

```
# Initialize dictionary with default value '0' for each key
keys = ['a', 'b', 'c']
defaultValue = 0

D = dict.fromkeys(keys,defaultValue)

print(D)
# Prints {'a': 0, 'b': 0, 'c': 0}
```

There is one more way to create a dictionary based on existing dictionary, called Dictionary comprehension.

Important Properties of a Dictionary

Dictionaries are pretty straightforward, but here are a few points you should be aware of when using them.

Keys must be unique:

A key can appear in a dictionary only once.

Even if you specify a key more than once during the creation of a dictionary, the last value for that key becomes the associated value.

```
D = {'name': 'Bob',
    'age': 25,
    'name': 'Jane'}
print(D)
# Prints {'name': 'Jane', 'age': 25}
```

Notice that the first occurrence of ‘name’ is replaced by the second one.

Key must be immutable type:

You can use any object of immutable type as dictionary keys – such as numbers, strings, booleans or tuples.

```
D = {(2,2): 25,
    True: 'a',
    'name': 'Bob'}
```

An exception is raised when mutable object is used as a key.

```
# TypeError: unhashable type: 'list'
D = {[2,2]: 25,
    'name': 'Bob'}
```

Value can be of any type:

There are no restrictions on dictionary values. A dictionary value can be any type of object and can appear in a dictionary multiple times.

```
# values of different datatypes
D = {'a':[1,2,3],
    'b':{1,2,3}}
```

```
# duplicate values
D = {'a':[1,2],
    'b':[1,2],
    'c':[1,2]}
```

Access Dictionary Items

The order of `key:value` pairs is not always the same. In fact, if you write the same example on another PC, you may get a different result. In general, the order of items in a dictionary is unpredictable.

But this is not a problem because the items of a dictionary are not indexed with integer indices. Instead, you use the keys to access the corresponding values.

You can fetch a value from a dictionary by referring to its key in square brackets `[]`.

```
D = {'name': 'Bob',
    'age': 25,
    'job': 'Dev'}

print(D['name'])
# Prints Bob
```

If you refer to a key that is not in the dictionary, you'll get an exception.

```
print(D['salary'])
# Triggers KeyError: 'salary'
```

To avoid such exception, you can use the special dictionary get() method. This method returns the value for key if key is in the dictionary, else `None`, so that this method never raises a `KeyError`.

```
# When key is present
print(D.get('name'))
# Prints Bob
```

```
# When key is absent
print(D.get('salary'))
# Prints None
```

Add or Update Dictionary Items

Adding or updating dictionary items is easy. Just refer to the item by its key and assign a value. If the key is already present in the dictionary, its value is replaced by the new one.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}

D['name'] = 'Sam'
print(D)
# Prints {'name': 'Sam', 'age': 25, 'job': 'Dev'}
```

If the key is new, it is added to the dictionary with its value.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}

D['city'] = 'New York'
print(D)
# Prints {'name': 'Bob', 'age': 25, 'job': 'Dev', 'city': 'New York'}
```

Merge Two Dictionaries

Use the built-in update() method to merge the keys and values of one dictionary into another. Note that this method blindly overwrites values of the same key if there's a clash.

```
D1 = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


D2 = {'age': 30,
     'city': 'New York',
     'email': 'bob@web.com'}


D1.update(D2)
print(D1)
# Prints {'name': 'Bob', 'age': 30, 'job': 'Dev',
#         'city': 'New York', 'email': 'bob@web.com'}
```

Remove Dictionary Items

There are several ways to remove items from a dictionary.

Remove an Item by Key

If you know the key of the item you want, you can use pop() method. It removes the key and returns its value.

```
D = {'name': 'Bob',
    'age': 25,
    'job': 'Dev'}


x = D.pop('age')
print(D)
# Prints {'name': 'Bob', 'job': 'Dev'}


# get removed value
print(x)
# Prints 25
```

If you don’t need the removed value, use the del statement.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


del D['age']

print(D)
# Prints {'name': 'Bob', 'job': 'Dev'}
```

Remove Last Inserted Item

The popitem() method removes and returns the last inserted item.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


x = D.popitem()

print(D)

# Prints {'name': 'Bob', 'age': 25}


# get removed pair

print(x)
# Prints ('job', 'Dev')
```

In versions before 3.7, popitem() would remove a random item.

Remove all Items

To delete all keys and values from a dictionary, use clear() method.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}
```

```
D.clear()

print(D)
# Prints {}
```

Get All Keys, Values and Key:Value Pairs

There are three dictionary methods that return all of the dictionary's keys, values and key-value pairs: keys(), values(), and items(). These methods are useful in loops that need to step through dictionary entries one by one.

All the three methods return iterable object. If you want a true list from these methods, wrap them in a `list()` function.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


# get all keys
print(list(D.keys()))
# Prints ['name', 'age', 'job']


# get all values
print(list(D.values()))
# Prints ['Bob', 25, 'Dev']


# get all pairs
print(list(D.items()))
# Prints [('name', 'Bob'), ('age', 25), ('job', 'Dev')]
```

Iterate Through a Dictionary

If you use a dictionary in a for loop, it traverses the keys of the dictionary by default.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


for x in D:
    print(x)
# Prints name age job
```

To iterate over the values of a dictionary, index from key to value inside the for loop.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


for x in D:
    print(D[x])
# Prints Bob 25 Dev
```

Check if a Key or Value Exists

If you want to know whether a key exists in a dictionary, use in and not in operators with if statement.

```
D = {'name': 'Bob',
     'age': 25,
     'job': 'Dev'}


print('name' in D)
# Prints True
print('salary' in D)
```

```
# Prints False
```

To check if a certain value exists in a dictionary, you can use method `values()`, which returns the values as a list, and then use the `in` operator.

```
D = {'name': 'Bob',
    'age': 25,
    'job': 'Dev'}


print('Bob' in D.values())
# Prints True
print('Sam' in D.values())
# Prints False
```

in Operator on List vs Dictionary

The `in` operator uses different algorithms for lists and dictionaries. For lists, it uses a search algorithm. As the list gets longer, the search time gets longer. For dictionaries, Python uses a different algorithm called Hash Table, which has a remarkable property: the operator takes the same amount of time, regardless of how many items are in the dictionary.

Find Dictionary Length

To find how many `key:value` pairs a dictionary has, use len() method.

```
D = {'name': 'Bob',
    'age': 25,
    'job': 'Dev'}


print(len(D))
# Prints 3
```

Python Dictionary Methods

Python has a set of built-in methods that you can invoke on dictionary objects.

Method	Description
clear()	Removes all items from the dictionary
copy()	Returns a shallow copy of the dictionary
fromkeys()	Creates a new dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list of key:value pair
keys()	Returns a list of all keys from dictionary
pop()	Removes and returns single dictionary item with specified key.
popitem()	Removes and returns last inserted key:value pair from the dictionary.
setdefault()	Returns the value of the specified key, if present. Else, inserts the key with a specified value.
update()	Updates the dictionary with the specified key:value pairs
values()	Returns a list of all values from dictionary

Built-in Functions with Dictionary

Python also has a set of built-in functions that you can use with dictionary objects.

Method	Description
all()	Returns True if all list items are true
any()	Returns True if any list item is true
len()	Returns the number of items in the list
sorted()	Returns a sorted list