1. Dictionaries

This is a composite data type, associative array of **key-value pairs** implemented as **hash tables**. **Values** can be **any Python objects**.

Dictionaries are similar to lists: **MUTABLE**, but **elements cannot be accessed by index** but via **keys** that **must be immutable** (i.e. **str**, **num**, **tuple**).

NOTE: Just as the values in a dictionary don't need to be of the same type, the keys don't either. Dictionaries can be nested as well. Lists and dictionaries are two of the most frequently used Python types. They have several similarities, but differ in how their elements are accessed. Lists elements are accessed by numerical index based on order, and dictionary elements are accessed by a key.

1 Defining

3 ways of how to define a dictionary.

Curly braces { }

Default syntax.

```
x_dict = {'Name': 'Var',
          'Mean':  121.3,
          'Meas_times': (1,2,3,4)}
```

dict(list of tuples)

```
MLB_team = dict([('Colorado', 'Rockies'),
                 ('Boston', 'Red Sox'),
                 ('Minnesota', 'Twins'),
                 ('Milwaukee', 'Brewers'),
                 ('Seattle', 'Mariners') ])
```

dict(key = value, ...)

```
MLB_team = dict(Colorado = 'Rockies',
                Boston = 'Red Sox',
                Minnesota = 'Twins',
                Milwaukee = 'Brewers',
                Seattle = 'Mariners')
```

2 Accessing values

Use respective **key in square brackets** [ ]. `KeyError` exception if this key not exists.

```
MLB_team['Minnesota']
> 'Twins'
```

Nothing in common with indexing:

```
d[0:2]
> TypeError: unhashable type: 'slice'
```

3 Adding/updating entry

```
MLB_team['Kansas City'] = 'Royals'
```

```
MLB_team['Kansas City'] = 'Royals_Updt'
```

Note: Although access to items in a dictionary does not depend on order, Python does guarantee that the order of items in a dictionary is preserved. When displayed, items will appear in the order they were defined, and iteration through the keys will occur in that order as well. Items added to a dictionary are added at the end. If items are deleted, the order of the remaining items is retained.

4 Deleting entry

```
del MLB_team['Seattle']
```

5 Building a dictionary incrementally

You can start by creating an empty dictionary, which is specified by empty curly braces. Then you can add new keys and values one at a time:

```
person = {}

person['fname'] = 'Joe'
person['age'] = 51
person['children'] = ['Ralph', 'Betty', 'Joey']
person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}
```

Retrieving the **values in the sublist or subdictionary** requires an **additional index or key**:

```
person['children'][0]
> 'Ralph'
```

6 Restrictions on keys

- Given key can appear in a dictionary only once. **Duplicate keys are not allowed**;
- **Dictionary key must be immutable**. **Integer**, **float**, **string**, and **bool** can be dictionary keys.

A **tuple** can also be a dictionary key, because tuples are also immutable:

```
d = {(1,1): 'a',
     (1,2):  0,
     (2,1): 'c',
     (2,2): 'd'}

d[(1,2)]
> 0
```

Note: **mutable = unhashable**. which means it can't be passed to a hash function. A hash function takes data of arbitrary size and maps it to a relatively simpler fixed-size value called a hash value (or simply hash), which is used for table lookup and comparison.

```
hash((1,2,3))
> 2528502973977326415

hash([1,2,3])
> TypeError: unhashable type: 'list'
```

7 Operators

```
in
not in
del
```

Tip: use operator `in` together with `and` to avoid raising an error if the key is not in the dictionary:

```
'Toronto' in MLB_team and MLB_team['Toronto']
# 'and' is a lazy operator: the second part will not be evaluated
> False
```

8 Methods

`get(key)` `clear()` `keys()` `values()` and items() then `pop(key)`, `popitem()` or `update(obj)`

```
d = {'a': 10, 'b': 20, 'c': 30}
```

`.get(key[,default])`
The **value behind a key** if it exists in the dictionary. If key is not found, it returns `None`.

```
d.get('b')
> 20

d.get('z', 'Try again, hooman!') # default option used
> 'Try again, hooman!'
```

`.clear()`
Empties dictionary entirely.

```
d.clear()
> {}
```

`.keys()`
A **list** of all keys in a dictionary.

```
d.keys()
> dict_keys(['a', 'b', 'c'])
```

`.values()`
Returns a **list** of all values in a dictionary.

```
d.values()
> dict_values([10, 20, 30])

list(d.values())[2]
> 30
```

`.items()`
A **list of tuples** containing key-value pairs in a dictionary.

```
d.items()
> dict_items([('a', 10), ('b', 20), ('c', 30)])

list(d.items())[0][1]
> 10
```

Note: The .items(), .keys(), and .values() methods return something called a view object. A dictionary view object is like a window on the keys and values. For practical purposes, you can think of these methods as returning lists of the dictionary's keys and values.

`.pop(key[, default])`
Removes a key from the dictionary, **and returns its value**.
KeyError exception if the key is not in a dict.

```
v = d.pop('b')
v
> {'a': 10, 'c': 30}
> 20

d.pop('z', 'Avoid KeyError exception, hooman!')  # default option used
> 'Avoid KeyError exception, hooman!'
```

`.popitem()`
Removes a random key-value pair from dict and returns it as a tuple.

```
d = {'a': 10, 'b': 20, 'c': 30}
v = d.popitem()

> {'a': 10, 'b': 20}
> ('c', 30)
```

`.update(obj)`
Merge a dict with another dict or with an iterable of key-value pairs.

```
d1 = {'a': 10, 'b': 20, 'c': 30}
d2 = {'b': 200, 'd': 400}

d1.update(d2) # note that keys are ordered alphabetically!
> {'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

`obj` may also be **a sequence of key-value pairs**. For example, `obj` can be specified as a **list of tuples**:

```
d1 = {'a': 10, 'b': 20, 'c': 30}

d1.update( [('b', 200), ('d', 400)] )
> {'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

9 Functions: same as for lists, except sum

```
max(d)
min(d)
len(d)
```

Good practice example for dict

Bad practice:

```
auth = None
if 'auth_token' in payload:
    auth = payload['auth_token']
else:
    auth = 'Unauthorized'
```

Good practice:

```
# use properties of .get() method for more idiomatic code
auth = payload.get('auth_token', 'Unauthorized')
```