Dictionaries

Collections of `key` - `value` pairs.

In [ ]:
```
my_empty_dict = {}  # alternative: my_empty_dict = dict()
print('dict: {}, type: {}'.format(my_empty_dict, type(my_empty_dict)))
```

Initialization

In [ ]:
```
dict1 = {'value1': 1.6, 'value2': 10, 'name': 'John Doe'}
dict2 = dict(value1=1.6, value2=10, name='John Doe')

print(dict1)
print(dict2)

print('equal: {}'.format(dict1 == dict2))
print('length: {}'.format(len(dict1)))
```

`dict.keys()`, `dict.values()`, `dict.items()`

In [ ]:
```
print('keys: {}'.format(dict1.keys()))
print('values: {}'.format(dict1.values()))
print('items: {}'.format(dict1.items()))
```

Accessing and setting values

In [ ]:
```
my_dict = {}
my_dict['key1'] = 'value1'
my_dict['key2'] = 99
my_dict['key1'] = 'new value'  # overriding existing value
print(my_dict)
print('value of key1: {}'.format(my_dict['key1']))
```

Accessing a nonexistent key will raise `KeyError` (see `dict.get()` for workaround):

In [ ]:
```
# print(my_dict['nope'])
```

Deleting

In [ ]:
```
my_dict = {'key1': 'value1', 'key2': 99, 'keyX': 'valueX'}
del my_dict['keyX']
print(my_dict)

# Usually better to make sure that the key exists (see also pop() and popitem())
key_to_delete = 'my_key'
if key_to_delete in my_dict:
    del my_dict[key_to_delete]
else:
    print('{key} is not in {dictionary}'.format(key=key_to_delete, dictionary=my_dict))
```

Dictionaries are mutable

In [ ]:
```
my_dict = {'ham': 'good', 'carrot': 'semi good'}
my_other_dict = my_dict
my_other_dict['carrot'] = 'super tasty'
my_other_dict['sausage'] = 'best ever'
print('my_dict: {}\nother: {}'.format(my_dict, my_other_dict))
print('equal: {}'.format(my_dict == my_other_dict))
```

Create a new `dict` if you want to have a copy:

In [ ]:
```
my_dict = {'ham': 'good', 'carrot': 'semi good'}
my_other_dict = dict(my_dict)
my_other_dict['beer'] = 'decent'
print('my_dict: {}\nother: {}'.format(my_dict, my_other_dict))
print('equal: {}'.format(my_dict == my_other_dict))
```

`dict.get()`

Returns `None` if `key` is not in `dict`. However, you can also specify `default` return value which will be returned if `key` is not present in the `dict`.

In [ ]:
```
my_dict = {'a': 1, 'b': 2, 'c': 3}
d = my_dict.get('d')
print('d: {}'.format(d))

d = my_dict.get('d', 'my default value')
print('d: {}'.format(d))
```

`dict.pop()`

In [ ]:
```
my_dict = dict(food='ham', drink='beer', sport='football')
print('dict before pops: {}'.format(my_dict))

food = my_dict.pop('food')
print('food: {}'.format(food))
print('dict after popping food: {}'.format(my_dict))

food_again = my_dict.pop('food', 'default value for food')
print('food again: {}'.format(food_again))
print('dict after popping food again: {}'.format(my_dict))
```

`dict.setdefault()`

Returns the `value` of `key` defined as first parameter. If the `key` is not present in the dict, adds `key` with default value (second parameter).

In [ ]:
```
my_dict = {'a': 1, 'b': 2, 'c': 3}
a = my_dict.setdefault('a', 'my default value')
d = my_dict.setdefault('d', 'my default value')
print('a: {}\nd: {}\nmy_dict: {}'.format(a, d, my_dict))
```

`dict.update()`

Merge two `dict`s

In [ ]:
```
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3}
dict1.update(dict2)
print(dict1)

# If they have same keys:
dict1.update({'c': 4})
print(dict1)
```

The keys of a `dict` have to be immutable

Thus you can not use e.g. a `list` or a `dict` as key because they are mutable types :

In [ ]:
```
# bad_dict = {['my_list'], 'value'}  # Raises TypeError
```

Values can be mutable

In [ ]:
```
good_dict = {'my key': ['Python', 'is', 'still', 'cool']}
print(good_dict)
```