8 Dictionary

1. A dictionary is a mapping

Dictionary

- A dictionary is like a list, but more general.
- In a list, there are values and corresponding indices. For example, in a list, `l = ['a', 'b', 'c']`, the values are 'a', 'b', and 'c', and the indices are 0, 1, 2, and 3.
- In a dictionary, the indices can be (almost) any type.
- A dictionary contains a collection of indices, which are called *keys*, and a collection of values. Each key is associated with a single value.
- The association of a key and a value is called *Key-value pair* or sometime an *item*.

Defining Dictionary

- We can *define a dictionary* in the following way.

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
```

- We can *add new items*

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
>>> eng2arabic['four'] = 4
>>> eng2arabic
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

- Define an empty dictionary, then add items.

```
>>> d = dict() # d would be an empty dictionary
>>> d['one'] = 1
>>> d['two'] = 2
>>> d
{'one': 1, 'two': 2}
```

2. Lookup

`len` function

- The `len` function works on dictionaries; it returns the number of key-value pairs:

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
>>> len(eng2arabic)
3
```

Lookup

- Then we can *access the values* via the keys.

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
>>> eng2arabic['one']
1
```

`in` operator

- The in operator works on dictionaries, too; it tells you whether something appears as a *key* in the dictionary.

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
>>> 'one' in eng2arabic
True
>>> 1 in eng2arabic
False
```

Get keys and values

- We can get keys and values using the following code.

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
>>> eng2arabic.keys()
dict_keys(['one', 'two', 'three'])
>>> eng2arabic.values()
dict_values([1, 2, 3])
```

- For sure, we can transform them into lists by using `list()` function

```
>>> eng2arabic = {'one': 1, 'two': 2, 'three': 3}
>>> list(eng2arabic.keys())
['one', 'two', 'three']
>>> list(eng2arabic.values())
[1, 2, 3]
```

- Then, we can print the pairs in the dictionary.

```
eng2arabic = {'one': 1, 'two': 2, 'three': 3}

for key in eng2arabic.keys():
    print(key, eng2arabic[key])
```

3. Dictionary and list/tuple

- We mentioned that tuples are useful for collecting data that are highly correlated.

```
cities = ['PISCATAWAY', 'HIGHLAND PARK', 'FLAGTOWN']
zipcodes = ['08854', '08904', '08821']

l_pairs = list(zip(cities, zipcodes))

print(l_pairs)
```

- The output would be as follows. It is a list with three tuples. In each tuple, there are city names and the corresponding zip-code.

```
[('PISCATAWAY', '08854'), ('HIGHLAND PARK', '08904'), ('FLAGTOWN',
'08821')]
```

- We can also use dictionary, which is more convenient for us to loop up.

```
cities = ['PISCATAWAY', 'HIGHLAND PARK', 'FLAGTOWN']
zipcodes = ['08854', '08904', '08821']

d = dict(zip(cities, zipcodes))
print(d)

print(d['PISCATAWAY'])
```

Dictionary comprehension

- Instead of `zip` function, we can also use dictionary comprehension just like list comprehension.

```
cities = ['PISCATAWAY', 'HIGHLAND PARK', 'FLAGTOWN']
zipcodes = ['08854', '08904', '08821']

d = {
    cities[i]: zipcodes[i]
    for i in range(len(cities))
}

print(d['PISCATAWAY'])
```

- It seems the result is the same as that by using `zip` function. But comprehension could be very useful in some scenarios. For example, in the following example, we have a list of student names and a list of ages of the corresponding students. We would like to get the dictionary for students older than 20 years old.

```
names = ['Jack', 'Mark', 'Mary', 'Jenny']
ages = [22, 18, 21, 17]

d = {
    names[i]: ages[i]
    for i in range(len(names))
    if ages[i]>20
}
print(d)
```

- The output would be

```
{'Jack': 22, 'Mary': 21}
```

4. Practice questions

1. Character counts. Given a string, try to obtain the counts of the characters in it (case insensitive). For example, given "Rutgers, RBS", the output should be the dictionary, {'T': 1, 'R': 3, 'B': 1, 'E': 1, ' ': 1, 'S': 2, 'U': 1, 'G': 1}.
2. Reverse lookup. The following dictionary shown the rosters of the three courses. We can easily check the student names each course has. Now we would like to do reverse lookup. That is, we would like to get a dictionary showing the courses that each student is taking.

```
roster = {
    'STATS_385': ['Jack', 'Anne', 'John', 'Peter'],
    'PROG_388': ['Tom', 'Mark', 'Jack', 'Jin'],
    'LARGESCALE_487': ['Anne', 'John', 'Jin', 'Jack']
}
```