- **created by**: Sudip Ghimire
- **URL**: https://www.sudipghimire.com.np
- **GitHub**: https://github.com/ghimiresdp

go to course contents

Chapter 3.3. Dictionary

Table of Contents

Introduction

We can visualize python dictionaries as regular dictionary book. If we want to find the meaning of the word `apple`, we search for the keyword `apple` and find the value associated to it.

Dictionaries in python are similar to lists but contains key-value pairs. We generally access items of the dictionary using the key instead of index.

Some of the features of dictionary are as follows:

- Dictionaries are similar to lists but represented by *key-value* pairs
- Dictionaries are ordered (since python 3.7)
- They are written inside *curley-brackets* or *braces* `{}`
- Keys must be unique otherwise the later one replaces the original one
- Dictionaries are mutable.
- Dictionaries can have multiple data types in their key-value pair

Structure/syntax

```
{key_1: value_1, key_2: value2, ...}

# multiline
{
    key_1: value_1,
    key_2: value_2,
    ...
        key_n: value_n
}
```

Creating a dictionary

We can create a dictionary by adding key value pairs within curley brackets.

Example 1:

```
dict_1 = {'a': 1, 'b': 2, 'c': 3}

# using type hinting for dictionary
dict_2: dict = {'a': 1, 'b': 2, 'c': 3}
```

Example 2:*multiline dictionary*

```
person = {
    'name': 'Spock',
    'age': 20,
    'married': False,
    'positions': ['Lieutenant', 'Captain', 'Commander'],
}
```

Example 3: *If we add the duplicate key, the second key-value pair replaces the original one*

```
person = {
    'name': 'Spock',
    'age': 20,
    'married': False,
    'age': 21,
}
print(person)

# Out: {'name': 'Spock', 'age': 21, 'married': False}
```

Getting values from the dictionary

Similar to lists or tuple, we can get values from the dictionary using keys inside the large brackets of the dictionary.

```
person = {
    'name': 'Spock',
    'age': 20,
    'married': False,
    'age': 21,
}

print(person['name'])       # Spock
print(person['married'])    # False
```

Sometimes we are not sure whether the key exists in the dictionary yet, in this case, we can use the `dict.get()` method to get the value from the key, or return the fallback value (the default value is `None`)

Example:

```
income = {
    'salary': 2000,
    'lease': 1000,
    'stock': 1200,
}

# print(salary['remittance'])       # KeyError
print(income.get('remittance'))     # None
print(income.get('remittance', 0))  # 0
```

Adding/updating an item to the dictionary

We can directly use Assignment operator to add/update values of the dictionary. To do so, we use the key of the dictionary inside the large brackets followed by `=` sign and then the value to assign/update the value.

```
income = {
    'salary': 2000,
    'lease': 1000,
    'stock': 1200,
}

income['freelancing'] = 700     # this adds new key `freelancing` to the dict
income['salary'] = 2200     # this updates the existing key `salary` to 2200
print(income)

# {'salary': 2200, 'lease': 1000, 'stock': 1200, 'freelancing': 700}
```

Adding/updating multiple items to the dictionary

We can add/update multiple items to the dictionary using update method, or using unpacking methods.
Example 1:

```
income = {
    'salary': 2000,
    'lease': 1000,
    'stock': 1200,
}

other_sources = {
    'uber': 600,            # new item
    'remittance': 700,      # new item
    'salary':2400           # existing item
}
income.update(other_sources)
print(income)

# {'salary': 2400, 'lease': 1000, 'stock': 1200, 'uber': 600, 'remittance': 700}
```

removing an item from the dictionary

We can use `pop()` method to remove an item from the dictionary. the `pop()` method takes one argument. If the argument is one of the keys of the dictionary, then it removes the item with that key. If the key does not exist in the dictionary, then it throws key error.

The `dict.pop()` method returns the value of the key that is supplied in the argument of the method.

```
income = {
    'salary': 2000,
    'lease': 1000,
    'stock': 1200,
    'uber': 600,
}

income.pop('salary')
# {'lease': 1000, 'stock': 1200, 'uber': 600}

value = income.pop('uber')
print(value)
# 600

income.pop('occupation')
# KeyError: 'occupation'
```

popping out an arbitrary item from the dictionary

We can use `dict.popitem()` to remove an arbitrary item from the dictionary. the `popitem()` method do not take any argument and returns both `key` and `value` as a tuple.

```python
income = {
    'salary': 2000,
    'lease': 1000,
    'stock': 1200,
    'uber': 600,
}

(key, value) = income.popitem()

print(key, value)
# uber 600
```

getting list of keys, values and items of the dictionary

We can use `dict.keys()`, `dict.values()`, and `dict.items()` to get the list-like iterables from the dictionary.

```python
person = {
    'name': 'Spock',
    'age': 20,
    'married': False,
}

print(person.keys())
# dict_keys(['name', 'age', 'married'])

print(person.values())
# dict_values(['Spock', 20, False])

print(person.items())
# dict_items([('name', 'Spock'), ('age', 20), ('married', False)])
```

Some useful dictionary methods

- `dict.copy()`
- `dict.fromkeys()`
- `dict.setdefault()`
- `dict.clear()`