LEARN

Dictionaries in Python in 4 Steps

01 **Definitions**

02 **Creating a Dictionary**

03 **Main Operations with Dictionaries**

04 **Nested Dictionaries**

Dictionaries

April 6, 2023

DICTIONARIES

1 . Definitions

- In this topic, we will examine the collection types which store item pairs. What does it mean?
- Think of a real dictionary. It contains words and their meanings. In Python, you can accept the words as key and the meaning of the words as value.
- A dictionary in Python is a collection of key-value pairs called items of a dictionary. The dictionary is enclosed by curly braces {}. Each pair (item) is separated by a comma and the key and value are separated by a colon.

2 . Creating a Dictionary

- A dictionary also can be created by enclosing pairs, separated by commas, in curly-braces.
- We can use a function to create a dictionary : 'dict()' function. Let's create a simple empty dict :

[3]:
```
empty_dict_1 = {}
empty_dict_2 = dict()
print(type(empty_dict_1),type(empty_dict_2))
```

```
<class 'dict'> <class 'dict'>
```

[4]:
```
#The basic form of dict looks like :
my_dict = {'key1': 'value1',
           'key2' : 'value2',
           'key3' : 'value3'}
```

- The syntax for accessing an item is very simple. We write a key that we want to access in square brackets. This method works both for adding items to a dict and for reading them from there.
- In the following examples, there are several methods that allow us to create a dict and add a key-value pair to it.

[5]:
```
state_capitals = {'Arkansas':'Little Rock',
                  'Colorado': 'Denver',
                  'California': 'Sacramento',
                  'Georgia': 'Atlanta'
                  }
```

```
print(state_capitals['Georgia']) #accessing method
```

```
Atlanta
```

[6]:
```
state_capitals = {'Arkansas':'Little Rock',
                  'Colorado': 'Denver',
                  'California': 'Sacramento',
                  'Georgia': 'Atlanta'
                  }
state_capitals ['Virginia'] = 'Richmond' # adding a new item
print(state_capitals)
```

```
{'Arkansas': 'Little Rock', 'Colorado': 'Denver', 'California': 'Sacramento',
'Georgia': 'Atlanta', 'Virginia': 'Richmond'}
```

- Keys and values can be of different types.

[7]:
```
mix_values = {'animal' : ('dog', 'cat'),  # tuple type
              'planet':['Neptun', 'Saturn', 'Jupiter'], #list type
              'number':40, # int type
              'pi': 3.14, # float type
              'is_good': True, # boolean type
              }
mix_keys = {22 : "integer",
            1.2 : "float",
            True : "boolean",
            "key" : "string"}
```

- Let's create a dictionary by using dict()

[8]:
```
dict_by_dict = dict(animal='dog', planet = 'neptun', number=40, pi=3.14,␣
 ↪is_good = True)
print(dict_by_dict)
```

```
{'animal': 'dog', 'planet': 'neptun', 'number': 40, 'pi': 3.14, 'is_good': True}
```

- Do not use quotes for keys when using the dict() function to create a dictionary.
- You cannot use iterables as keys to create a dictionary.

3 . Main Operations with Dictionaries

- There are several methods that allow us to access items, keys, and values. You can access all items using the .items() method, all keys using the .keys() method, and all values using the .values() method:

[10]:
```
dict_by_dict = {'animal':'dog',
                'planet':'neptun',
                'number':40,
                'pi':3.14,
```

```
                'is_good':True}
print(dict_by_dict.items(), '\n')
print(dict_by_dict.keys(), '\n')
print(dict_by_dict.values(), '\n')
```

```
dict_items([('animal', 'dog'), ('planet', 'neptun'), ('number', 40), ('pi',
3.14), ('is_good', True)])

dict_keys(['animal', 'planet', 'number', 'pi', 'is_good'])

dict_values(['dog', 'neptun', 40, 3.14, True])
```

- You have learned that you can add a new item by assigning value to a key that is not in the dictionary. Likewise, you can add new items using the .update() method.

[11]:
```
dict_by_dict = {'animal':'dog',
                'planet':'neptun',
                'number':40,
                'pi':3.14,
                'is_good':True}
dict_by_dict.update({'is_bad':False})
print(dict_by_dict)
```

```
{'animal': 'dog', 'planet': 'neptun', 'number': 40, 'pi': 3.14, 'is_good': True,
'is_bad': False}
```

- You can also remove an item using the del function:

[12]:
```
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True,
                'is_bad': False}
del dict_by_dict['animal']
print(dict_by_dict)
```

```
{'planet': 'neptun', 'number': 40, 'pi': 3.14, 'is_good': True, 'is_bad': False}
```

- Using the in and the not in operator, you can check if the key is in the dictionary.

When we use the in operator; if the key is in the dictionary, the result will be True otherwise False.

When we use the not in; if the key is not in the dictionary, the result will be True otherwise False.

[13]:
```
dict_by_dict = {'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True,
```

```
               'is_bad': False}
print('pi' in dict_by_dict)
print('animal' not in dict_by_dict) # remember , we have deleted 'animal'
```

```
True
True
```

4 . Nested Dictionaries

[14]:
```
school_records={
    "personal_info":
    {"kid":{"tom":{"class":"intermediate","age": 10},
            "sue": {"class": "elementary","age": 8}},
    "teen":{"joseph":{"class":"college","age":19},
            "marry":{"class":"high school","age":16}},},
            "greades_info":
            {"kid":{"tom":{"math":88, "speech":69},
                    "sue":{"math":90, "speech":81}},
                    "teen":{"joseph":{"coding":80, "math":89},
                            "marry":{"coding":70, "math":96}},},
}
```

- We can use square brackets to access internal dicts :

[15]:
```
school_records={
    "personal_info":
        {"kid":{"tom": {"class":"intermediate", "age":10},
                "sue": {"class":"elementary", "age":8}
               },
         "teen":{"joseph":{"class":"college", "age":19},
                 "marry":{"class":"high school", "age":16}
                },
        },
}
print(school_records['personal_info']['teen']['marry']['age'])
```

```
16
```

- Dictionaries strongly resemble JSON syntax. The native json module in the Python standard library can be used to convert between JSON and dictionaries.

If you want to go deep into dicts, https://l24.im/wvJ you will find what you want.