Chapter-7 Python Dictionaries

Contents

Chapter-7 Python Dictionaries

List နှင့် tuple တို့တွင် sequence များ အကြောင်းကို လေ့လာခဲ့ပြီးနောက် mapping လုပ်ခြင်း အကြောင်း ဆက်လက်လေ့လာကြအောင်။ Sequence ဆိုသည်မှာ object များကို ၎င်းတို့၏ နေရာအစဉ်(by their relative position)ဖြင့် တွဲ၍ သိမ်းဆည်း(store)ထားခြင်း ဖြစ်သည်။ (sequence stored objects by their relative position)။

Mapping ဆိုသည်မှာ object များကို ၎င်းတို့၏ key နှင့်တွဲ၍ သိမ်းဆည်းထားသည်။ Mapping လုပ်သည့် dictionary တွင် နေရာသည် အရေးကြီးသည့် အရာမဟုတ်ပေ။ Python dictionary တစ်ခုတွင် key နှင့် သက်ဆိုင်ရာ(associated) value တွဲထားသည့် အတွဲများ ပါဝင်သည်။ တခြားသော programming language များ လေ့လာခဲ့ဘူးပါက dictionary ကို hash table, associative array, hashe, hashmap များ အဖြစ် မှတ်ယူနိုင်သည်။

Python dictionary သည် list နှင့် ခပ်ဆင်ဆင် တူညီသည်။ List တွင် နေရာအစဉ်(position order)သည် အဓိက ကြသည်။ Dictionary တွင် key သည် အဓိက ကြသည်။ Key သည် များသောအားဖြင့် string အမျိုးအစား ဖြစ်သည်။ သို့သော် boolean, integer, float, tuple, string စသည့်အမျိုးအစား immutable item များ ဖြစ်နိုင်သည်။

7.1 Dictionary ဆိုတာ ဘာလဲ?

Dictionary သည် ဒေတာများကို သိမ်းဆည် ထားနိုင်သည့် data structure အမျိုးအစား တစ်ခု ဖြစ်သည်။ ဥပမာ - လူတစ်ဦး၏ အမည်နှင့် အသက်အရွယ်ကို structure အဖြစ် ဖွဲ့စည်းတည်ဆောက်၍ dictionary ပုံစံဖြင့် သိမ်းဆည်းထားနိုင်သည်။

Dictionary ထဲတွင် ဒေတာများကို **key-value pair** ပုံစံမျိုးဖြင့် သိမ်းဆည်းသည်။ Dictionary ထဲတွင် သိမ်းဆည်းသည့် ဒေတာများသည် နေရာအစီအစဉ်တကျ သိုလှောင်ထားခြင်းမရှိပါ။ (Dictionaries do not store information in any particular order) ။ ထို့ကြောင့် သင်ထည့်သွင်းသည့် အစဉ်အတိုင်း ဒေတာများကို ပြန်လည် ရရှိလိမ့်မည်မဟုတ်ပါ။ Dictionary တွင် ပါရှိသည့် item များသည် key-value pair များ ဖြစ်ကြသည်။

List တွင် ဒေတာများကို index နံပါတ် ဖြင့် access လုပ်သည်။ Dictionary တွင် ဒေတာများကို key ဖြင့် access လုပ်ခြင်းကြောင့် key သည် မှတ်မိရန် ပိုလွယ်ကူသည်။

Dictionary ထဲတွင် ဒေတာများကို order သို့မဟုတ် position နှင့် မသိမ်းဆည်းထားသောကြောင့် indexing လုပ်သည့်အခါ index နံပါတ်များ ထည့်ပေး၍ မရပါ။ Dictionary တွင် index လုပ်သည့်အခါ key ထည့်ပေး ရပါသည်။ အဘယ်ကြောင့်ဆိုသော် ဒေတာများကို key-value pair ပုံစံမျိုးဖြင့် သိမ်းဆည်းသောကြောင့် ဖြစ်သည်။

Python dictionary ကို အတိုခေါက်အားဖြင့် dict ဟူ၍လည်းခေါ်လေ့ရှိသည်။ Unordered dictionary (dict) နှင့် ordered dictionary ဟူ၍ dictionary နှစ်မျိုး ရှိသည်။ Unordered dictionary(dict) သည် default ဖြစ်သည်။ OrderedDict တွင် key-value pair ၏ ထည့်ပေးသည့် အစဉ်(order of insertion)ကို မှတ်ထားသည်။ ဒေတာများကို key-value pair ပုံစံမျိုးဖြင့် အောက်ပါအတိုင်း သိမ်းဆည်းသည်။

```
dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3}
```

တွန့်ကွင်း (curly brackets {}) အတွင်း၌ **key : value pair** များကို ကော်မာဖြင့် ခွဲရေးပြီး(comma-separated) ထည့်သွင်းထားခြင်းဖြင့် dictionary တည်ဆောက်နိုင်သည်။ အောက်ပါအတိုင်း ရေးလျှင်ပို၍ ဖတ်ရ လွယ်ကူပြီး နားလည် သဘောပေါက်နိုင်သည်။ Python ၏ list, tuple, dictionary စသည်တို့ အတွင်းရှိ item များကို ရေးသည့်အခါ နောက်ဆုံးအလုံး၏ နောက်တွင် comma ချန်ထားခဲ့ပြီး ရေးနိုင်သည်။

```
dictionary_name = {
"key_1": "value_1",
"key_2": "value_2",
"key_3": "value_3",
}
```

7.2 Constructing a Dictionary

Dictionary တည်ဆောက်ခြင်း နည်းနှစ်မျိုးဖြင့် dictionary များကို တည်ဆောက်နိုင်သည်။

(၁) တွန့်ကွင်း(curly brackets { }) ထဲတွင် key-value pair များကို ကော်မာခံ(comma)ရေး၍ Python dictionary တည်ဆောက်နိုင်သည်။ key-value pair ကို ရေးသည့်အခါ key : value ပုံစံမျိုးဖြင့် ရေးသားသည်။

(၂) **dict()** function ကို သုံး၍ တည်ဆောက်နိုင်သည်။

```
empty_dict = {}      #empty dictionary
empty_dict
```

```
{}
```

အောက်တွင် **dict()** function ဖြင့် dictionary တည်ဆောက်နိုင်သည်။ Variable d ဖြင့် တည်ဆောက်ထားသည့် dictionary တစ်ခုကို assign လုပ်သည်။ **dict()** function ဖြင့် dictionary တည်ဆောက်သည့်အခါ assignment operator(=) ကိုသုံး၍ key နှင့် value များကို တွဲပေးရသည်။ State နှင့် city သည် key များ ဖြစ်ကြသည် ။ NY နှင့် New York တို့သည် key နှင့်သက်ဆိုင်သည့် value များ ဖြစ်ကြသည်။

```
d = dict(state="NY", city="New York")
d
```

```
{'state': 'NY', 'city': 'New York'}
```

key : value ပုံစံမျိုးဖြင့် ရေးသည်။

```
my_dict = {'key1':'value1','key2':'value2' }
print(my_dict)
```

#တည်ဆောက်လိုက်သည့် dictionary ၏ အမျိုးအစားကို **type()** function စစ်နိုင်သည်။

```
print(type(my_dict))
```

```
{'key1': 'value1', 'key2': 'value2'}
<class 'dict'>
```

တွန့်ကွင်း (curly brackets { }) ထဲတွင် key-value pair များကို ကော်မာခံ(comma)ရေး၍ thisdict ဆိုသည့် dictionary တစ်ခု တည်ဆောက်သည်။

```
thisdict = {
  "brand": "Honda",
  "model": "HRV",
  "year": 2017
}
print(thisdict)
```

```
{'brand': 'Honda', 'model': 'HRV', 'year': 2017}
```

"model" ဆိုသည့် key ကို အသုံးပြု၍ ထို key နှင့် တွဲထားသည့် value ကို ထုတ်ယူသည်။

```
x = thisdict["model"]
x
```

```
'HRV'
```

get() method ကို အသုံးပြုနိုင်သည်။ အဖြေတူသည်။

```
x = thisdict.get("model")      # Get the value of the "model" key:
x
```

```
'HRV'
```

Dictionary အတွင်းရှိ value များကို ပြောင်းနိုင်သည်။ (Change Values)။ Key name ကို အသုံးပြု၍ ၄င်းနှင့်သက်ဆိုင်သည့် value ကို ပြောင်းနိုင်သည်။ "year" ဆိုသည့် key ဖြင့် value ဖြစ်သည့် 2017 ကို 2018 ဖြစ်အောင် ပြောင်းသည်။

```
car_dict={"brand": "Honda", "model": "HRV", "year": 2017}
car_dict["year"] = 2018                # Change the "year" to 2018
car_dict
```

```
{'brand': 'Honda', 'model': 'HRV', 'year': 2018}
```

7.2.1 Convert by Using dict()

List of two-item lists ကို dictionary တစ်ခု အဖြစ် ပြောင်းခြင်း

dict() function ကို အသုံးပြု၍ value နှစ်ခု ပါသည့် ကိန်းအစဉ်အတန်း(two-value sequences)ကို dictionary တစ်ခု အဖြစ် ပြောင်းနိုင်သည်။

```
# example using lol (a list of two-item lists):
```

```
lol=[['a', 'b'], ['c', 'd'], ['e', 'f']]
dict(lol)
```

```
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Two-item Tuple ကို Dictionary တစ်ခုအဖြစ် ပြောင်းခြင်း

အောက်တွင် two-item tuples တစ်ခုကို dictionary တစ်ခုအဖြစ် ပြောင်းပုံကို ဥပမာအဖြစ် ဖော်ပြ ထားသည်။

```
lot = [('a', 'b'), ('c', 'd'), ('e', 'f')]
dict(lot)
```

```
{'a': 'b', 'c': 'd', 'e': 'f'}
```

List of Two-Character Strings တစ်ခုကို Dictionary တစ်ခုအဖြစ် ပြောင်းခြင်း

အောက်တွင် two-character strings list တစ်ခုကို dictionary တစ်ခုအဖြစ် ပြောင်းပုံကို ဥပမာ အဖြစ် ဖော်ပြထားသည်။

```
los=['ab', 'cd', 'ef']
dict(los)
```

```
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Dictionary ထဲတွင် key များ၏ အစဉ်(order of keys)သည် ကြိုက်သလို(arbitrary) ဖြစ်နိုင်သည်ကို သတိပြုပါ။ Dictionary အတွင်းရှိ key များကို တစ်ခုချင်းစီ ဖော်ပြပါ။

```
car_dict = {"brand": "Ford", "model": "Mustang", "year": 1964}
for x in car_dict:
    print(x)                # key များကို for loop ဖြင့် ပတ်ပြီး ထုတ်ယူသည်။
```

```
brand
model
year
```

Dictionary အတွင်းရှိ key နှင့် value များကို တွဲ၍ တစ်ခုချင်းစီ ဖော်ပြပါ။

```
car_dict = {"brand": "Ford", "model": "Mustang", "year": 1964}
for x in car_dict:
    print(x,":" ,thisdict[x], )        # key နှင့် value များကို ထုတ်ယူသည်။
```

```
brand : Ford
model : Mustang
year : 1964
```

7.3 [key] ဖြင့် item များ အသစ်ထည့်ခြင်း၊ ရှိပြီးသားများကိုပြောင်းခြင်း

key ကို အသုံးပြု၍ dictionary ထဲသို့ item အသစ်များ ထည့်ခြင်း၊ ရှိပြီးသား item များကို ပြောင်းခြင်း

```
dictionary_name = {"key_1": "value_1", "key_2": "value_2",
                   "key_3": "value_3"}
dictionary_name
```

```
{'key_1': 'value_1', 'key_2': 'value_2', 'key_3': 'value_3'}
```

[key] ကို သုံး၍ "key_4" နှင့် "value_4" ကို key-value pair အသစ် အဖြစ် ထည့်သည်။

```
dictionary_name["key_4"]="value_4"   # key_4 နှင့် value_4 ကို ထည့်သည်။
print(dictionary_name)
```

```
{'key_1': 'value_1', 'key_2': 'value_2', 'key_3': 'value_3',
'key_4': 'value_4'}
```

Key-value pair တစ်ခုကို dictionary သို့ထည့်သည့်အခါ dictionary ထဲတွင် key သည် ရှိပြီးသား ဖြစ်လျှင် ရှိပြီးသား(existing) value ကို value အသစ်ဖြင့် ပြောင်းလဲ အစားထိုးသည်။ Dictionary ထဲတွင် key မရှိလျှင် ထည့်ပေးသည်။ ဤအချက်သည် list နှင့် မတူသည့်အချက်တစ်ခု ဖြစ်သည်။ Dictionary တစ်ခု အတွင်း၌ တစ်ခုနှင့် တစ်ခု မတူညီသည့် key များသာ ရှိရမည်။ (Dictionary keys must be unique.)။ တူညီသည့် key နှစ်ခု မရှိနေစေရ။

[key] ကို သုံး၍ "value_2" ကို "value_TWO" အဖြစ်သို့ ပြောင်းသည်။

```
dictionary_name["key_2" ] = "value_TWO"
print(dictionary_name)
```

```
{'key_1': 'value_1', 'key_2': 'value_TWO', 'key_3': 'value_3',
'key_4': 'value_4'}
```

7.4 update() ဖြင့် Dictionary နှစ်ခု ပေါင်းခြင်း

update() function ကို သုံး၍ dictionary တစ်ခုမှ key-value pair များကို တခြား dictionary တစ်ခုထဲသို့ ကော်ပီကူးထည့်နိုင်သည်။ အောက်တွင် `dict_1.update(dict_2)` ဖြင့် dict_2 မှ key-value pair များကို dict_1 သို့ ကော်ပီကူး ထည့်သည်။

```
dict_1={'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
dict_2={'key_4':'value_4', 'key_5':'value_5', 'key_6':'value_6'}
dict_1.update(dict_2)
print(dict_1)
```

```
{'key_1': 'value_1', 'key_2': 'value_2', 'key_3': 'value_3',
'key_4': 'value_4', 'key_5': 'value_5', 'key_6': 'value_6'}
```

ဒီလိုမျိုး **update()** နဲ့ dictionary နှစ်ခုကို ပေါင်း(merge)တဲ့အခါ dictionary နှစ်ခု စလုံးမှာ တူညီတဲ့ key တွေ ပါရှိနေရင်ဘယ်လို လုပ်မလဲ? ဒုတိယ(နောက်က) dictionary ကို အနိုင်ပေးပါသည်။ ဥပမာ- ဒုတိယ(နောက်က) dictionary က key 'b' ရဲ့ value 'platypus' ကိုပဲ update လုပ်သည်။

```
first = {'a': 1, 'b': 2}
second = {'b': 'platypus'}
```

```
first.update(second)
first
```

```
{'a': 1, 'b': 'platypus'}
```

7.5 del() ဖြင့် key-value pair တစ်ခုကို ဖျက်ခြင်း

del key word ကိုသုံး၍ key နာမည်ဖြင့် dictionary ထဲမှ item ကို ဖျက်(delete)နိုင်သည်။

```
dict_1={'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
del dict_1['key_1']
dict_1
```

```
{'key_2': 'value_2', 'key_3': 'value_3'}
```

Dictionary ထဲမှ value များ ရယူရန် (access လုပ်ရန်) **values()** method ကို အသုံးပြုနိုင်သည်။

```
sales={'apple':2, 'orange':3, 'grapes':4}
print(sales.values())
```

```
dict_values([2, 3, 4])
```

7.6 clear() ဖြင့် Dictionary ထဲမှ Key များ အားလုံးကို ဖျက်ခြင်း

Dictionary ထဲမှ key များ အားလုံးနှင့် value များ အားလုံးကို ဖျက်လိုလျှင် **clear()** ကို သုံးနိုင်သည် သို့မဟုတ် empty dictionary ({}) တစ်ခုဖြင့် ဖျက်မည့် dictionary ကို reassign အလုပ်နိုင်သည်။

```
dict_1.clear()
dict_1
```

```
{}
```

```
dict_1= {}        # Empty dictionary ({}) ဖြင့် dictionary name ကို reassign အလုပ်နိုင်သည်။
dict_1
```

```
{}
```

7.7 in Keyword ကိုသုံး၍ မိမိသိလိုသည့် key တစ်ခု ရှိ မရှိ စစ်ခြင်း

Dictionary ထဲတွင် မိမိသိလိုသည့် key တစ်ခု ရှိ မရှိ(exist or not) သိလိုလျှင် **in** key wordကို သုံး၍ စစ်နိုင်သည်။

```
dict_1={'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
'key_1' in dict_1
```

```
True
```

```
'key_5' in dict_1
```

```
False
```

```
car_dict = {"brand": "Ford", "model": "Mustang", "year": 1964}
```

```
if "model" in car_dict:
  print("Yes,'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

7.8 [key] ဖြင့် Dictionary ထဲမှ Item တစ်ခုကို ထုတ်ယူခြင်း

ဤနည်းသည် dictionary တွင် အသုံးအများဆုံး ဖြစ်သည်။ Key ထည့်ပေး၍ ၎င်းနှင့် သက်ဆိုင်သည့် (corresponding) value ကို ရယူနိုင်သည်။

```
dict_1={'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
dict_1['key_1']
```

```
'value_1'
```

Dictionary ထဲတွင် မိမိ ထည့်ပေးသည့် key မရှိပါက exception (KeyError) ပြန်ပေးလိမ့်မည်။

```
dict_1['key_5']                          # KeyError ပေးလိမ့်မည်။
```

```
KeyError: 'key_5'
```

ထိုသို့ exception ပြန်ပေးခြင်း မဖြစ်ပေါ်စေရန် မိမိ ထည့်ပေးမည့် key ရှိ မရှိ စစ်နိုင်သည်။ သို့မဟုတ် **get()** function ကို အသုံးပြု နိုင်သည်။

```
dict_1.get('key_1')
```

```
'value_1'
```

key ကို အသုံးပြု၍ value များကို access လုပ်နိုင်သည်။ Dictionary က သိမ်းဆည်းနိုင်သည့် ဒေတာအမျိုးအစားများသည် ပြောင်းလွယ်ပြင်လွယ်(flexible)ဖြစ်သည်။ ဥပမာ-

```
my_dict={'key1':123,'key2':[12,23,33],
         'key3':['item0','item1','item2']}

my_dict['key2']            # Call values by their key
```

```
[12, 23, 33]
```

```
my_dict['key3']            # Let's call items from the dictionary
```

```
['item0', 'item1', 'item2']
```

`my_dict['key3'][0]` တွင် [0] သည် 'key3' ၏ value ဖြစ်သည့် list ၏ item နံပါတ် 0 ဖြစ်သည်။

```
my_dict['key3'][0]
```

```
'item0'
```

.upper() ဖြင့် အကြီး စာလုံး ပြောင်းသည်။

```
my_dict['key3'][0].upper()
```

```
'ITEM0'
```

```
my_dict['key1']
```

```
123
```

Dictionary ထဲမှ value များကို အောက်ပါကဲ့သို့ ပေါင်းခြင်း နုတ်ခြင်း မြှောက်ခြင်း စသည်တို့ပြုလုပ်ရန် built-in method ပါရှိသည်။ ဥပမာ- 'key1' နှင့် သက်ဆိုင်သည့် value မှ 123 ကို နုတ်(subtract)သည်။

```
my_dict['key1'] = my_dict['key1'] - 123  #(123- 123)
my_dict['key1']
```

```
0
```

```
# Set the object equal to itself minus 123
my_dict['key1'] -= 123                    #(0- 123)
my_dict['key1']
```

```
-123
```

7.8.1 keys() ဖြင့် Dictionary ထဲမှ Item အားလုံးကို ထုတ်ယူခြင်း

Dictionary ထဲမှ key အားလုံးကို တပြိုင်နက် access လုပ်ရန် **key()** ကို သုံးနိုင်သည်။ ဥပမာ-

```
dict_1={'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
dict_1.keys()
```

```
dict_keys(['key_1', 'key_2', 'key_3'])
```

list() function ကို သုံး၍ dictionary ထဲမှ key များ ၊ value များနှင့် item များကို list အဖြစ်သို့ ပြောင်းနိုင်သည်။ key() သည် item များ အားလုံးကို list တစ်ခု အဖြစ် ပြောင်းသည်။

```
dict_1={'key_1':'value_1', 'key_2':'value_2', 'key_3':'value_3'}
list(dict_1.keys())
```

```
['key_1', 'key_2', 'key_3']
```

7.9 values() ဖြင့် Dictionary ထဲမှ value အားလုံးကို ထုတ်ယူခြင်း

Dictionary ထဲမှ value အားလုံးကို တပြိုင်နက် ရယူရန် **values()** function ကို သုံးသည်။ Dictionary ထဲမှ value များအားလုံးကို list တစ်ခု အဖြစ် ထုတ်ယူသည်။

```
list(dict_1.values())
```

```
['value_1', 'value_2', 'value_3']
```

```
car_dict={"brand": "Ford", "model": "Mustang", "year": 1964}
for x in car_dict.values():
       print(x)
```

```
Ford
Mustang
```

```
1964
```

7.10 items() ဖြင့် Dictionary ထဲမှ key-value pairs အားလုံးကို ထုတ်ယူခြင်း

Dictionary ထဲမှ key-value pair အားလုံးကို အတွဲလိုက် ရယူရန် **items()** function ကိုသုံးသည်။ Dictionary ထဲမှ key-value pair များအားလုံးကို list တစ်ခု အဖြစ် ထုတ်ယူသည်။

```
list(dict_1.items())              # list အဖြစ် ထုတ်ယူသည်။
```

```
[('key_1', 'value_1'), ('key_2', 'value_2'), ('key_3', 'value_3')]
```

Dictionary ထဲမှ key-value pair များအားလုံးကို list တစ်ခု အဖြစ် မဟုတ်ဘဲ ထုတ်ယူသည်။

```
dict_1.items()                    # dict_items အဖြစ် ထုတ်ယူသည်။
```

```
dict_items([('key_1', 'value_1'), ('key_2', 'value_2'), ('key_3',
'value_3')])
```

```
car_dict={"brand":"Ford", "model":"Mustang", "year":1964}
for x, y in car_dict.items():
    print(x, ":" ,y)
```

```
brand : Ford
model : Mustang
year : 1964
```

7.11 Assignment operator(=) and copy()

Assignment operator(**=**) ဖြင့် assign လုပ်သည်။ **copy()** ဖြင့် ကော်ပီကူးသည်။

```
signals = {'green': 'go', 'yellow': 'go faster', 'red':
           'smile for the camera'}
save_signals = signals          # = operator ဖြင့် assign လုပ်သည်။
print(save_signals)
signals['blue'] = 'confuse everyone'
print(save_signals)
```

```
{'green': 'go', 'yellow': 'go faster', 'red': 'smile for the
camera'}
{'green': 'go', 'yellow': 'go faster', 'red': 'smile for the
camera', 'blue': 'confuse everyone'}
```

Dictionary တစ်ခုအတွင်းမှ key-value pair များကို တခြား dictionary တစ်ခုအဖြစ် ကော်ပီကူးလိုလျှင် copy()ကို သုံးသည်။ copy()ကို သုံးလျှင် မူလ dictionary နှင့် ကော်ပီကူးလိုက်သည့် dictionary ဟူ၍ သီးခြား dictionary (၂)ခု အဖြစ် တည်ရှိသည်။

```
dict_1={'key_1':'value_1','key_2':'value_2', 'key_3':'value_3'}
dict_2={'key_4':'value_4','key_5':'value_5', 'key_6':'value_6'}
```

```
dict_2 = dict_1.copy()
print(dict_1)
print(dict_2)
```

```
{'key_1': 'value_1', 'key_2': 'value_2', 'key_3': 'value_3'}
{'key_1': 'value_1', 'key_2': 'value_2', 'key_3': 'value_3'}
```

7.12 Assign လုပ်၍ key အသစ်များ ထပ်ထည့်ခြင်း

Key နှင့် value ကို တစ်ခုချင်းစီ သီးခြားထည့်သည့်နည်း ဖြစ်သည်။

```
# Create a new dictionary
d = {}
# assign လုပ်၍ key အသစ်များ ထပ်ထည့်ခြင်း (Create a new key through assignme
nt)
d['animal'] = 'Dog'

# Can do this with any object
d['answer'] = 42
d
```

```
{'animal': 'Dog', 'answer': 42}
```

7.13 Nesting with Dictionaries

Dictionary တစ်ခုထဲတွင် တခြားသော dictionary များ အလွှာလိုက် အထပ်ထပ် ရှိနေနိုင်သည်။ ထိုသို့ dictionary တစ်ခုထဲတွင် တခြားသော dictionary များ ရှိနေခြင်းကြောင့် Nested dictionary ဟု ခေါ်ဆိုခြင်း ဖြစ်သည်။

```
# Dictionary nested inside a dictionary nested inside a dictionary
d = {'key1':{'nestkey':{'subnestkey':'value'}}}
d
```

```
{'key1': {'nestkey': {'subnestkey': 'value'}}}
```

Nested dictionary ထဲမှ value များ မည်ကဲ့သို့ ထုတ်ယူနိုင်သည်ကို အောက်တွင် ဖော်ပြထားသည်။ Main dictionary မှ ['key1'] ၊ ဒုတိယ dictionary မှ ['nestkey'] ၊ တတိယအဆင့် dictionary မှ ['subnestkey'] စသည်ဖြင့် key များကို အဆင့်ဆင့် ထည့်ပေးရသည်။

```
# Keep calling the keys
d['key1']['nestkey']['subnestkey']
```

```
'value'
```

Nested dictionary အောက်တွင် dictionary တစ်ခုတွင် sub dictionary (၃)ခု ပါဝင်သည့် Nested dictionary တစ်ခုကို ဥပမာအဖြစ် ဖော်ပြထားသည်။

```
myfamily = {
  "child1": {"name" : "Emil", "year" : 2004},
```

```
  "child2": {"name" : "Tobias","year" : 2007},
  "child3": {"name" : "Linus","year" : 2011}
   }
myfamily
```

```
{'child1': {'name': 'Emil', 'year': 2004},
 'child2': {'name': 'Tobias', 'year': 2007},
 'child3': {'name': 'Linus', 'year': 2011}}
```

7.14 Dictionary Methods

Dictionary တစ်ခု တည်ဆောက်သည်။ **.keys()** ဖြင့် dictionary အတွင်းရှိ key များ အားလုံးကို ကြည့်နိုင်သည်။ ဖော်ပြနိုင်သည်။

```
d = {'key1':1,'key2':2,'key3':3}
d.keys()
```

```
dict_keys(['key1', 'key2', 'key3'])
```

.values() ဖြင့် dictionary အတွင်းရှိ value များ အားလုံးကို ကြည့်နိုင်သည်။ ဖော်ပြနိုင်သည်။

```
d.values()
```

```
dict_values([1, 2, 3])
```

.items() ဖြင့် အတွင်းရှိ item များ အားလုံး ဖော်ပြနိုင်သည်။ ကြည့်နိုင်သည်။ Dictionary တွင် ပါရှိသည့် item များသည် key-value pair များ ဖြစ်ကြသည်။

```
d.items()
```

```
dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```

Built-in **isinstance()** function ကိုသုံး၍ dictionary သည် dict class ၏ instance တစ်ခု ဖြစ်ကြောင်း စစ်ဆေးနိုင်သည်။

```
isinstance(dictionary, dict)
```

```
True
```

7.14 Dictionary တစ်ခုအတွင်းသို့ ဒေတာများထည့်ခြင်း

dict() function ကိုသုံး၍ dictionary တစ်ခုအတွင်းသို့ ဒေတာများကို ထည့်နိုင်သည်။ dict() function ကို သုံးသည့်အခါ assignment operator (=) ဖြင့် key များ value များကို တွဲ၍ assign လုပ်သည်။

```
dictionary1 = dict(state="NY", city="New York")
print(dictionary1)
```

```
{'state': 'NY', 'city': 'New York'}
```

{ } ကို သုံးလျှင် key နှင့် value အကြားတွင် : ခံ၍ ရေးသည်။

```
dictionary2 = {"state": "Maryland", "city": "Baltimore"}
print(dictionary2)
```

```
{'state': 'Maryland', 'city': 'Baltimore'}
```

တည်ဆောက်ပြီးသား တစ်ခုထဲသို့ key ကိုသုံး၍ value ကို ထည့်ပေး(assign လုပ်) နိုင်သည်။ (2. Assign values to existing dictionaries using keys)။ ဥပမာ- အထက်က ထဲသို့ 'bird': 'Baltimore oriole'ကို ထည့်သည်။ Key သည် 'bird' ဖြစ်သည်။ Value သည် 'Baltimore oriole' ဖြစ်သည်။

```
dictionary2['bird'] = 'Baltimore oriole'
dictionary2
```

```
{'state':'Maryland', 'city':'Baltimore', 'bird':'Baltimore oriole'}
```

7.15 Dictionary တစ်ခုမှ Data များကို ဖတ်ယူခြင်း (Reading Data from a Dictionary)

Dictionary မှ value များကို key ဖြင့် ဖတ်ယူနိုင်သည်။

```
print(dictionary1['state'])
```

```
NY
```

ထိုသို့ ဖတ်ယူသည့်အခါ ထည့်ပေးသည့် key သည် dictionary ထဲတွင် မရှိလျှင် KeyError ပေးလိမ့်မည်။

```
print(dictionary1['age'])
```

```
KeyError: 'age'
```

KeyError မဖြစ်ပေါ်စေရန်အတွက် **get()** function ကို အသုံးပြုသည်။ **get()** function ဖြင့် ဖတ်သည့်အခါ ထည့်ပေးသည့် key မရှိပါက None ပြန်ပေးသည်။ KeyError ဖြစ်ပေါ်ခြင်းကို ကျော်လွှား နိုင်သည်။

```
print(dictionary1.get('age'))
```

```
None
```

7.16 Iterating Through Dictionaries

Dictionary အတွင်းရှိ key နှင့် value များ တစ်တွဲချင်းစီကို iterate လုပ်ခြင်း for loop ကို သုံး၍ dictionary အတွင်းရှိ key နှင့် value များ တစ်တွဲချင်းစီကို iterate လုပ်ခြင်းသည် အရိုးရှင်း ဆုံးနည်းဖြစ်သည်။

```
dictionary1 = dict(state="NY", city="New York")
# for loop ဖြင့် get dictionary's keys ကို iterate လုပ်ခြင်း
for item in dictionary1:
    print(item)
```

```
state
city
```

Dictionary ထဲရှိ key သို့မဟုတ် value သို့မဟုတ် နှစ်မျိုးစလုံးကို iterate လုပ်နိုင်သည်။ **keys()** method ကို သုံး၍ key များကို ထုတ်ယူနိုင်သည်။ **values()** method ကို သုံး၍ value များကို ထုတ်ယူနိုင်သည်။

```
dictionary1 = dict( state="NY", city="New York")
for item in dictionary1.keys(): print(item)
```

```
state
city
```

```
dictionary1 = {'state': 'NY', 'city': 'New York'}
for item in dictionary1.values(): print(item)
```

```
NY
New York
```

keys နှင့် values နှစ်မျိုးစလုံးကို တစ်ပြိုင်နက် iterate လုပ်နိုင်သည်။ ထုတ်ယူနိုင်သည်။

```
dictionary1 = {'state': 'NY', 'city': 'New York'}
for keys, values in dictionary1.items():
       print(keys, ':', values)
```

```
state : NY
city : New York
```

7.17 Checking for the Existence of Particular Keys

for loop ကို သုံး၍ dictionary အတွင်း၌ iterating လုပ်စရာမလိုဘဲ **in** keyword ကို သုံး၍ မိမိသိလိုသည့် key တစ်ခုသည် dictionary ထဲတွင် ရှိမရှိကို စစ်နိုင်သည်။ ရှိလျှင် True(Boolean value) သို့မဟုတ် မရှိလျှင် False (Boolean value) ကို ပြန်ပေးလိမ့်မည်။

```
a={"size": "10 feet", "weight": "16 pounds"}
print("size" in a)
print("length" in a)
```

```
True
False
```

7.18 Additional Dictionary Attributes

Dictionary တစ်ခုပြုလုပ်ပြီးသည့်နှင့် တပြိုက်နက် dictionary object တစ်ခု ပြုလုပ်ပြီး ဖြစ်သည်။ Dictionary object တွင် ရှိသည့် attributes နှင့် properties တို့ကို built-in **dir()** function သုံး၍ attribute များကို ကြည့်နိုင်သည်။

```
print(dir(a))
```

```
     ['__class__', '__contains__', '__delattr__', '__delitem__', '__
dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute
__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subcla
```

```
ss__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__
', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__seti
tem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy'
, 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault'
, 'update', 'values']
```

အောက်တွင် method တချို့ကို ဥပမာဖြင့် ရှင်းပြထားသည်။

.update() method

Dictionary ထဲသို့ key-value pair အသစ်များ ထည့်လိုသည့်အခါ .update() method ကိုသုံးသည်။ Empty dictionary တစ်ခုထဲသို့ **.update()** method ဖြင့် key-value pair အသစ်တစ်ခု ထည့်ပုံကို ဥပမာအဖြစ် ပြထားသည်။

```
a = {}
a.update({"name": "Dan Brown"})   # key-value pair အသစ်တစ်ခု ထည့်သည်။
a
```

```
{'name': 'Dan Brown'}
```

ရှိပြီးသား key ဖြစ်လျှင် ထို key-value pair များကို ထပ်မထည့်ပေးပါ။ ထို့ကြောင့် **.update()** function သည် dictionary နှစ်ခုကို ပေါင်း(merge)ပြီး မတူညီသည့်(different) key များကို ရလိုသည့်အခါ အသုံးဝင်သည်။ ရှိပြီးသား key-value pair ကို ထပ် မထည့်ပေးသောကြောင့် ဖြစ်သည်။

clear()

Dictionary တစ်ခုအတွင်းမှ key များ အားလုံးကို ဖယ်ရှား ရှင်းထုတ်လိုသည့်အခါ clear() method ကို သုံးသည်။

```
a = {'name': 'Dan Brown Xavier'}
a.clear()
a
```

```
{}
```

del keyword

key-value pair တစ်ခုတည်းကို ဖယ်ထုတ်လိုလျှင် **del** keyword ကို သုံးသည်။

```
a = {"name": "Skandar Keynes", "age": "24"}
del a["name"]
a
```

```
{'age': '24'}
```

pop()

Dictionary ထဲမှ key-value pair ကို ဖယ်ထုတ်ပြီး ထိုဖယ်ထုတ်လိုက်သည့် key-value pair ကို ကိစ္စတစ်ခုခု အတွက် အသုံးလိုလျှင် pop() ကိုသုံးသည်။ **pop()** သည် ထိုဖယ်ထုတ်လိုက်သည့် key-value pair ကို သိမ်းဆည်းထားသည်။

```
a = {"name": "Skandar Keynes", "age": "24"}
b = a.pop("name")
print(a)
print(b)
```

```
{'age': '24'}
Skandar Keynes
```

.copy() method

Shallow copy အဖြစ် ကော်ပီ ကူးလိုသည့်အခါ **copy()** method ကို သုံးသည်။ ကော်ပီကူးပြီးသည့်အခါ မူလ ဒေတာနှင့် ဘာမှ မသက်ဆိုင်တော့သည့် object အသစ်များ ဖြစ်သွားသည်။

```
a = {"name": "Skandar Keynes", "age": "24"}
b = a.copy()
print(a)
print(b)

a["name"]="Janet Jackson"   # a ထဲမှ "name"ကို "Janet Jackson" ဟု ပြောင်းသည်။
print(a)
print(b)      # b သည် a ၏ shallow copy ဖြစ်သည်။ a ပြောင်းသော်လည်း b လိုက် မပြောင်းပါ။
```

```
{'name': 'Skandar Keynes', 'age': '24'}
{'name': 'Skandar Keynes', 'age': '24'}
{'name': 'Janet Jackson', 'age': '24'}
{'name': 'Skandar Keynes', 'age': '24'}
```

အထက်က ဥပမာတွင် b သည် a ၏ shallow copy ဖြစ်သည်။ Dictionary a မှ key-value pair အားလုံးကို dictionary b အဖြစ် ကော်ပီကူးပေးပြီး အသစ်တစ်ခု ပြလုပ်ပေးသည်။ a ကို update လုပ်သည့်အခါ b သည် ဘာမှ မပြောင်းလဲပါ။ သီးခြားလွတ်လပ်သည့် dictionary နှစ်ခုအဖြစ် ရှိနေလိမ့်မည်။ မူလ dictionary a ပြောင်းသော် လည်း b လိုက်မပြောင်းပါ။

Deep Copy

Assignment operator(=) ကို အသုံးပြုပါက deep copy အဖြစ် ကော်ပီကူးပေးသည်။ a နှင့် b တို့သည် တူညီသည့် ဒေတာများကို ရည်ညွှန်းသည်။ ထို့ကြောင့် တစ်ခုကို update လုပ်လျှင် တခြားတစ်ခုသည် လိုက်၍ ပြောင်းလဲ လိမ့်မည်။ Assignment operator(=) သည် ရှိပြီးသား dictionary တစ်ခုကို နောက်ထပ် နာမည်တစ်ခု ထပ်ပေးခြင်း ဖြစ်သည်။

```
a = {"name": "Skandar Keynes", "age": "24"}
```

```
b = a
a["name"] = "Janet Jackson"
b["age"] = 16
print(a)
print(b)
```

```
{'name': 'Janet Jackson', 'age': 16}
{'name': 'Janet Jackson', 'age': 16}
```

popitem() method

popitem() method သည် dictionary ထဲမှ random item ထုတ်ယူပေးသည်။ ထိုသို့ထုတ်ယူပေးပြီးနောက် dictionary ထဲမှ ဖျက်လိုက်သည်။

```
a = {"name": "Skandar Keynes", "age": "24", "sex": "male"}
a.popitem()
```

```
('sex', 'male')
```

```
a
```

```
{'name': 'Skandar Keynes', 'age': '24'}
```

```
a = {"name": "Skandar Keynes", "age": "24", "sex": "male"}
b = a.setdefault("name")
a
```

```
{'name': 'Skandar Keynes', 'age': '24', 'sex': 'male'}
```

```
b
```

```
'Skandar Keynes'
```

7.19 Ordered Dictionaries

သာမာန် dictionary သည် ထည့်ပေးသည့် ဒေတာများ၏ အစဉ်ကို သိမ်းဆည်းထားလေ့ မရှိပေ။ (do not maintain the insertion order of the key-value pairs)။ ဒေတာများ၏ ထည့်ပေးသည့် အစဉ်(insertion order of keys)ကို ထိန်းသိမ်းထားလိုပါက သို့မဟုတ် ထည့်ထားပေးသည့် အစဉ်အတိုင်း dictionary ထဲတွင် ရှိနေစေလိုပါက ordered dictionary ကို အသုံးပြုရသည်။ (Ordered dictionaries are dictionaries that maintain the insertion order of keys.)။ Ordered dictionary ကို iterate လုပ်သည့်အခါ ထည့်ပေးသည့် အစဉ် အတိုင်း key များကို access လုပ်နိုင်သည်။

7.20 ChainMap() ကို အသုံးပြု၍ Dictionary နှစ်ခုဆက်ခြင်း

```
import collections
a = {"a": "A", "b": "B"}
b = {"c": "C", "d": "D"}
chained_dicts = collections.ChainMap(a, b)
```

```
print("Calling individual values from {}".format(chained_dicts))
```

```
Calling individual values from ChainMap({'a': 'A', 'b': 'B'}, {'c':
'C', 'd': 'D'})
```

```
print("Keys: ")
for key in chained_dicts.keys():
    print(key, end=" ")
```

```
Keys:
c d a b
```

```
print("Values: ")
for val in chained_dicts.values():
    print(val, end=" ")
```

```
Values:
C D A B
```

```
print("Key and Values")
for key, val in chained_dicts.items():
    print("{} = {}".format(key, val))
```

```
Key and Values
c = C
d = D
a = A
b = B
```

End