RAJALAKSHMI ENGINEERING COLLEGE

OCS1903-PROGRAMMING USING PYTHON

Dictionary

CONTENTS

- Dictionary
- Dictionary Creation
- Operators and Methods
- Examples

Towards Dictionaries

- Lists, tuples, and strings hold elements with only integer indices

45	"Coding"	4.5	7	89
0	1	2	3	4

Integer Indices

- In essence, each element has an index (or a key) which can only be an integer, and a value which can be of any type (e.g., in the above list/tuple, the first element has key 0 and value 45)
  - What if we want to store elements with non-integer indices (or keys)?

Dictionaries {}

- In Python, you can use a dictionary to store elements with keys of any hashable types (e.g., integers, floats, Booleans, strings, and tuples; but not lists and dictionaries themselves) and values of any types

45	"Coding"	4.5	7	89
"NUM"	**1000**	**2000**	**3.4**	**"XXX"**

keys of different types

Values of different types

- The above dictionary can be defined in Python as follows:

```
dic = {"NUM":45, 1000:"coding", 2000:4.5, 3.4:7, "XXX":89}
```

key ***value***

Each element is a ***key:value*** pair, and elements are separated by commas

Dictionaries {}

- Can contain any and different types of elements (i.e., hashable keys & values)
- Can contain only unique keys but duplicate values

```
dic2 = {"a":1, "a":2, "b":2}
print(dic2)
```

Output: {'a': 2, 'b': 2}

The element “a”:2 will override the element “a”:1 because only ONE element can have key “a”

- Can be indexed but only through keys (i.e., dic2[“a”] will return 1 but dic2[0] will return an error since there is no element with key 0 in dic2)

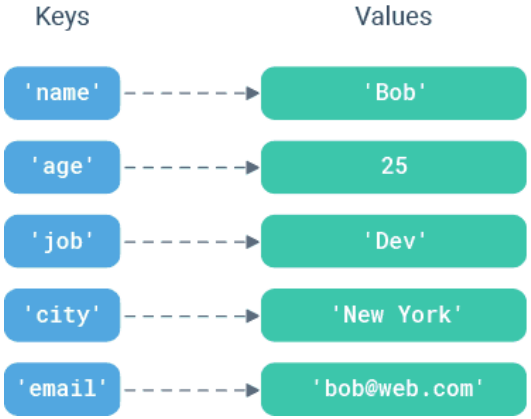
Dictionary-{}

- Dictionary is a composite data type in python similar to list.
- A dictionary consists of a collection of key-value pairs.
- Each key-value pair maps the key to its associated value.
- Dictionary is **mutable.**
- One Dictionary can be nested within another Dictionary.
- Can't be concatenated.
- As list elements are accessed through their index, **Dictionary elements are accessed via keys.**

Dictionary-{} Creation

Dictionary encloses comma separated list of key :value pairs.

```
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```




```
>>> d={'name':'Bob',
       'age':25,
       'job':'dev',
       'city':'New York',
       'email':'bob@web.com'}
```

Dictionary-{} Creation

```
d = dict([
    (<key>, <value>),
    (<key>, <value),
      .
      .
      .
    (<key>, <value>)
])
```

Empty dictionary

```
>>> d=dict()
>>> d
{}
```

List of tuples converted to dictionary

```
>>> d1=dict([(1,"Sorna"),(2,"Ida"),(3,"Vijay")])
>>> d1
{1: 'Sorna', 2: 'Ida', 3: 'Vijay'}
```

String as arguments,converted to dictionary

```
>>> d2=dict(name="Sorna",age=25)
>>> d2
{'name': 'Sorna', 'age': 25}
```

Dictionary-{} Creation

Building a Dictionary Incrementally

```
>>> person = {}
>>> type(person)
<class 'dict'>

>>> person['fname'] = 'Joe'
>>> person['lname'] = 'Fonebone'
>>> person['age'] = 51
>>> person['spouse'] = 'Edna'
>>> person['children'] = ['Ralph', 'Betty', 'Joey']
>>> person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}
```

```
>>> person
{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> person['fname']
'Joe'
>>> person['age']
51
>>> person['children']
['Ralph', 'Betty', 'Joey']
```

```
>>> person['children'][-1]
'Joey'
>>> person['pets']['cat']
'Sox'
```

Restrictions that dictionary keys must abide

Almost any type of value can be used as a dictionary key in Python . However, there are a couple restrictions that dictionary keys must abide by.

- **key can appear in a dictionary only once.**
- **Duplicate keys are not allowed.**
- **If you specify a key a second time during the initial creation of a dictionary, the second occurrence will override the first**
- **A dictionary key must be of a type that is immutable.**

- Can be iterated or looped over

```
dic = {"first": 1, "second": 2, "third": 3}
for i in dic:
    print(i)
```

Output:

first
second
third

ONLY the keys will be returned.

How to get the values?

Dictionaries {}

- Can be iterated or looped over

```
dic = {"first": 1, "second": 2, "third": 3}
for i in dic:
    print(dic[i])
```

Output:

1
2
3

Values can be accessed via indexing!

Operators and Built-in Functions

len() function can be used to find the no of key:value pairs in the dictionary.

Operators: **Membership operator** can be used in dictionary

```
>>> person
{'fname': 'Sorna', 'age': 39, 'spouse': 'Nesa', 'children': ['karthik', '
thula']}
>>> len(person)
4
>>> "fname" in person
True
>>> "Sorna" in person
False
```

Membership operator is used for keys alone

Dictionary Functions

Function	Description
dic.clear()	Removes all the elements from dictionary dic
dic.copy()	Returns a copy of dictionary dic
dic.items()	Returns a list containing a tuple for each key-value pair in dictionary dic
dic.get(k)	Returns the value of the specified key k from dictionary dic
dic.keys()	Returns a list containing all the keys of dictionary dic
dic.pop(k)	Removes the element with the specified key k from dictionary dic
dict.popitem()	Similar to get(),but will set dic[key]=default if key is not already in dic.

clear()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> d.clear()
>>> d
{}
```

get()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> print(d.get('b'))
20
>>> print(d.get('z'))
None
```

items()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.items())
[('a', 10), ('b', 20), ('c', 30)]
>>> list(d.items())[1][0]
'b'
>>> list(d.items())[1][1]
20
```

keys()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.keys())
['a', 'b', 'c']
```

values()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}

>>> list(d.values())
[10, 20, 30]
```

pop()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.pop('b')
20
>>> d
{'a': 10, 'c': 30}
```

popitem()

```
>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.popitem()
('c', 30)
>>> d
{'a': 10, 'b': 20}

>>> d.popitem()
('b', 20)
>>> d
{'a': 10}
```

update()

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d2 = {'b': 200, 'd': 400}

>>> d1.update(d2)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d1.update([('b', 200), ('d', 400)])
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d1.update(b=200, d=400)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

Example 1

Write a Python program to find sum, maximum and minimum values of the values of a dictionary and print the dictionary

Input:
{“Apple”:88,”Orange”:90,”Banana”:78}
Output:
Sum=256
Max:90
Min:78

```
marks=dict()
i=1
while(i!=-99):
    key=input()
    value=int(input())
    marks1=dict([(key,value)])
    marks.update(marks1)
    i=int(input("Enter -99 to finish dict creation"))
print(marks)
print("The sum is",sum(marks.values()))
print("The max value is",max(marks.values()))
print("The min value is",min(marks.values()))
for m in marks:
   print(marks[m])
```

Example 2

Write a Python program to create a dictionary from a string.

```
from collections import defaultdict, Counter
str1 = 'w3resource'
my_dict = {}
for letter in str1:
    my_dict[letter] = my_dict.get(letter, 0) + 1
print(my_dict)
```

Sample Output:

```
{'w': 1, '3': 1, 'r': 2, 'e': 2, 's': 1, 'o': 1, 'u': 1, 'c': 1}
```

Example 3

Write a Python program to concatenate following dictionaries to create a new one.

```
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
dic4 = {}
for d in (dic1, dic2, dic3): dic4.update(d)
print(dic4)
```

Sample Output:

```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```


THANK
YOU