Dictionaries

Agenda

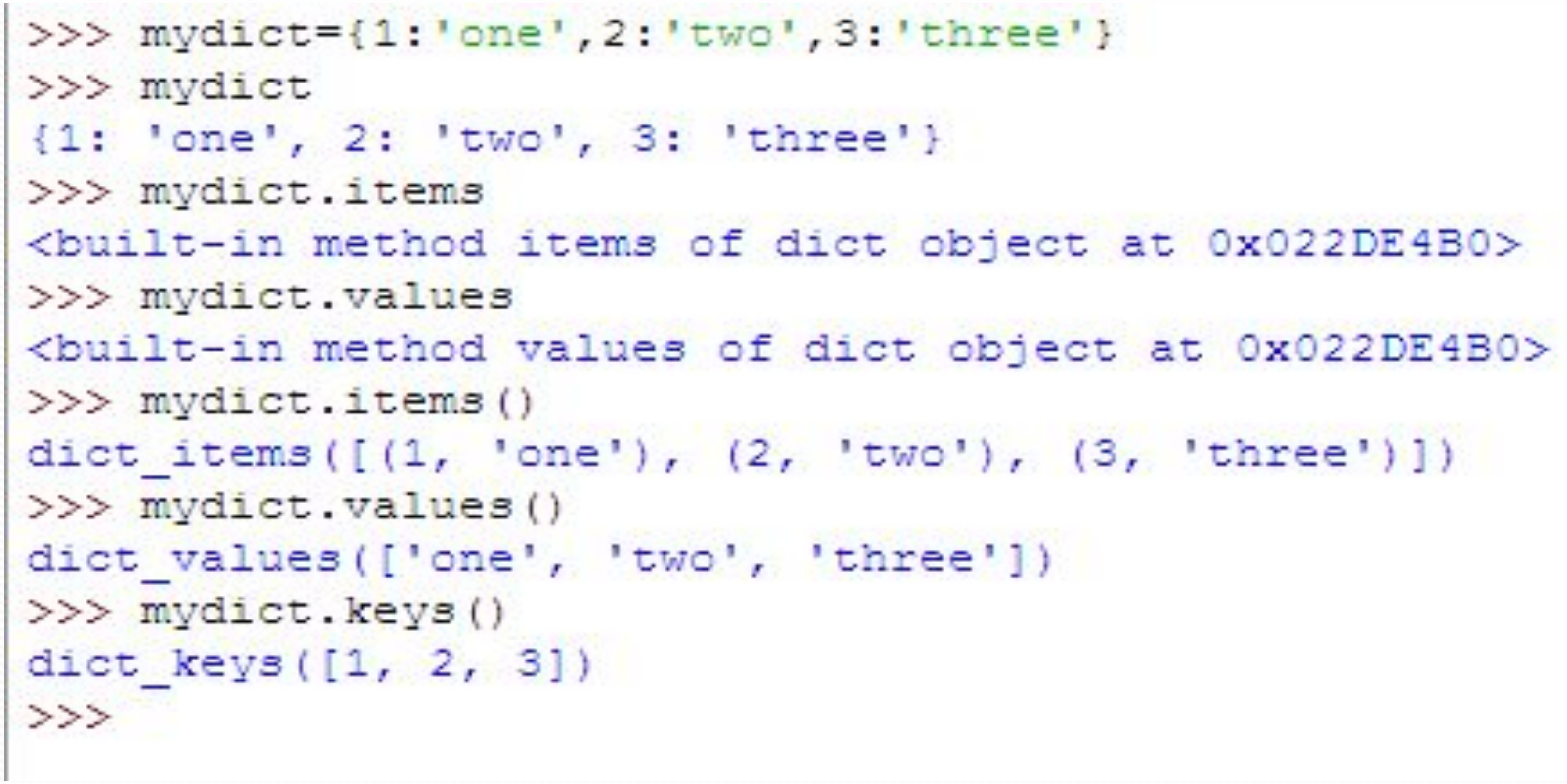
- What is Dictionary
- Getting values from Dictionary
- Adding Elements into Dictionary
- Removing keys form Dictionary
- Dictionary Methods
- Dictionary Comprehension

What is Dictionary

- **Python dictionary** is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.
- Dictionary are mutable
- **Syntax:** my_dict ={ }

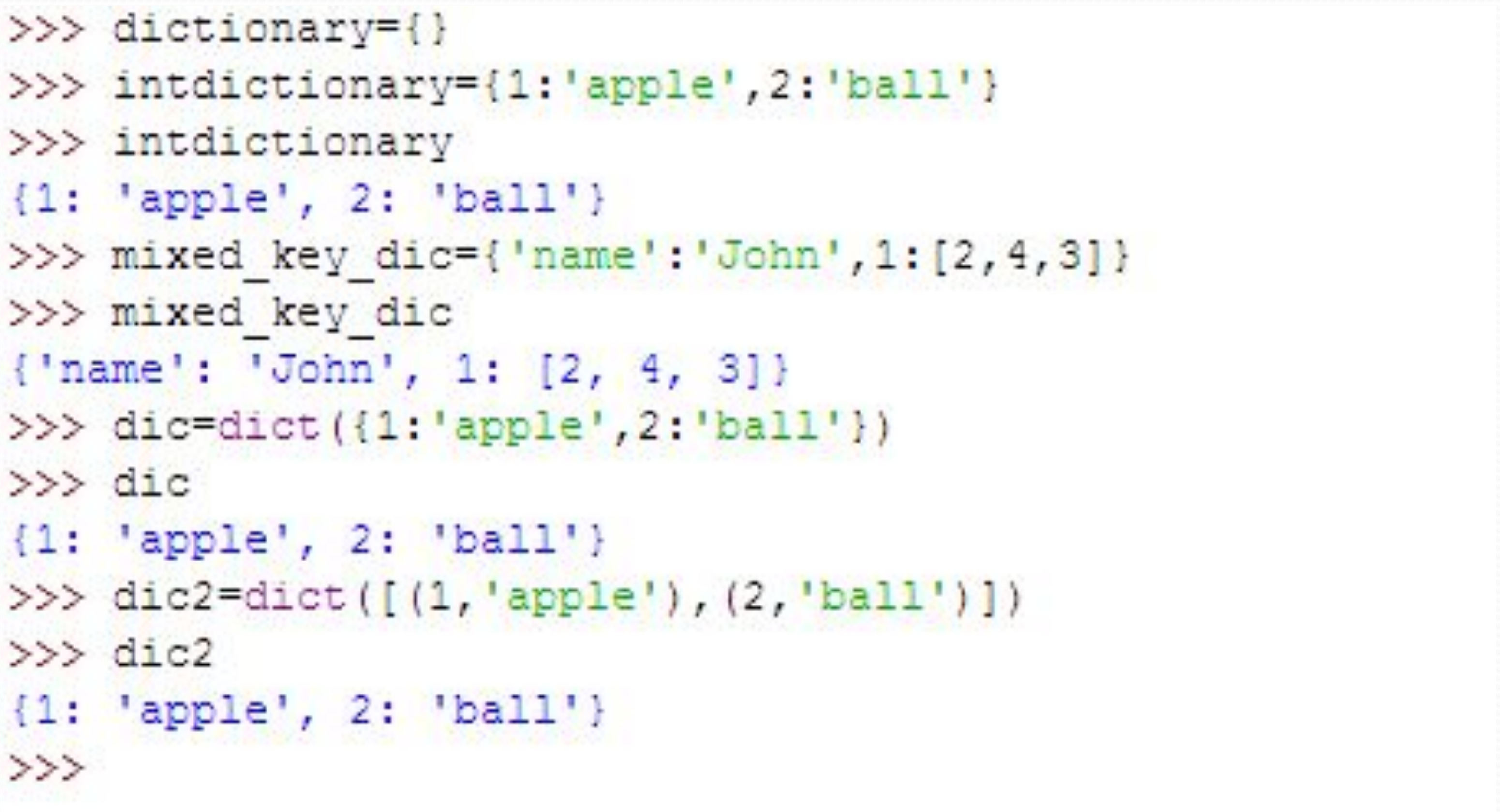
Dictionaries

- Python dictionary is an unordered collection of items.



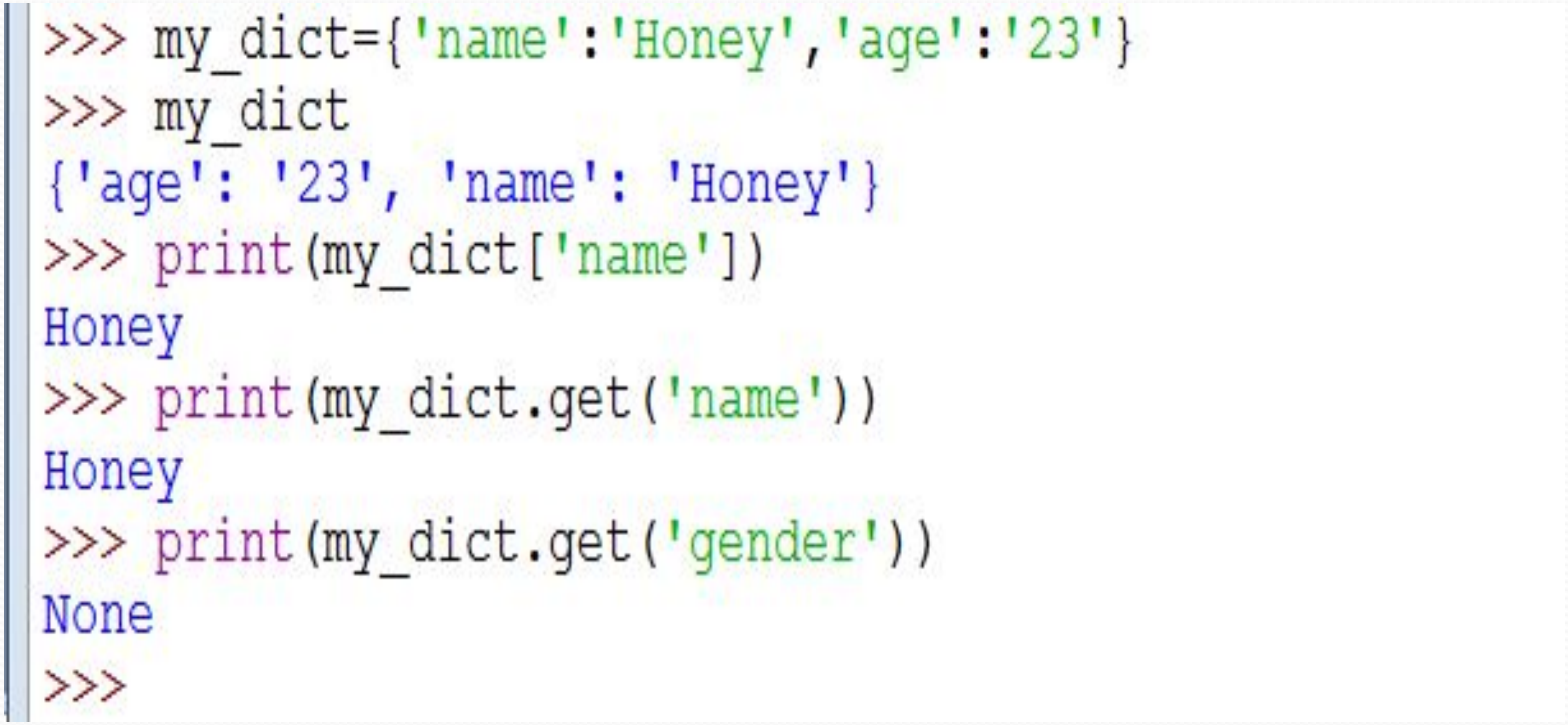
```
>>> mydict={1:'one',2:'two',3:'three'}
>>> mydict
{1: 'one', 2: 'two', 3: 'three'}
>>> mydict.items
<built-in method items of dict object at 0x022DE4B0>
>>> mydict.values
<built-in method values of dict object at 0x022DE4B0>
>>> mydict.items()
dict_items([(1, 'one'), (2, 'two'), (3, 'three')])
>>> mydict.values()
dict_values(['one', 'two', 'three'])
>>> mydict.keys()
dict_keys([1, 2, 3])
>>>
```

Continue…..



```
>>> dictionary={}
>>> intdictionary={1:'apple',2:'ball'}
>>> intdictionary
{1: 'apple', 2: 'ball'}
>>> mixed_key_dic={'name':'John',1:[2,4,3]}
>>> mixed_key_dic
{'name': 'John', 1: [2, 4, 3]}
>>> dic=dict({1:'apple',2:'ball'})
>>> dic
{1: 'apple', 2: 'ball'}
>>> dic2=dict([(1,'apple'),(2,'ball')])
>>> dic2
{1: 'apple', 2: 'ball'}
>>>
```

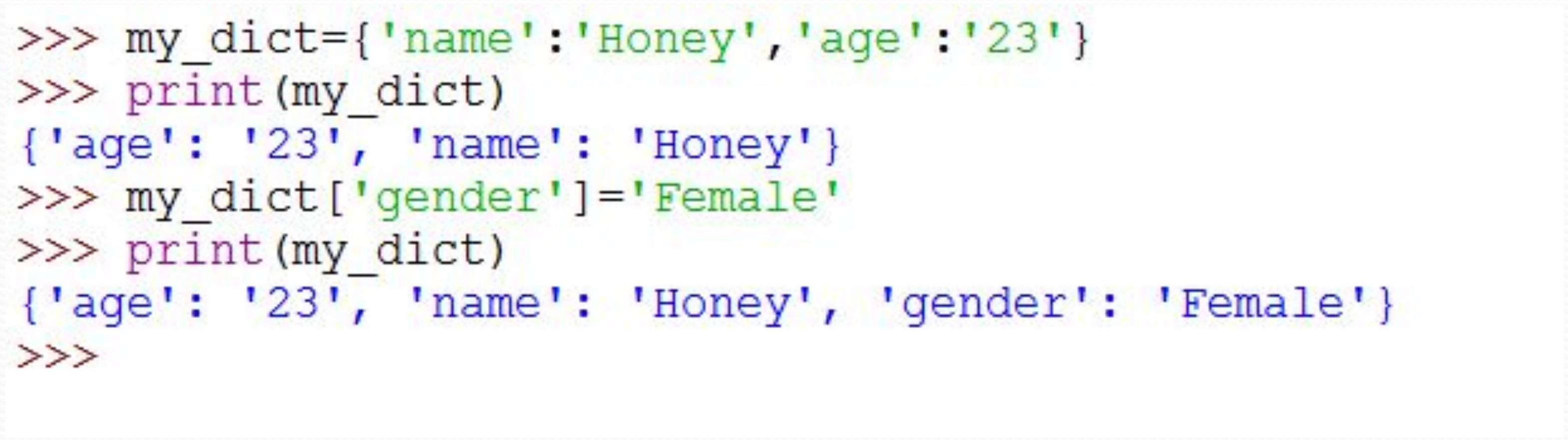
Getting Values from Dictionary



```
>>> my_dict={'name':'Honey','age':'23'}
>>> my_dict
{'age': '23', 'name': 'Honey'}
>>> print(my_dict['name'])
Honey
>>> print(my_dict.get('name'))
Honey
>>> print(my_dict.get('gender'))
None
>>>
```

Adding Elements into Dictionary

- Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.
- If the key is already present, value gets updated, else a new key: value pair is added to the dictionary

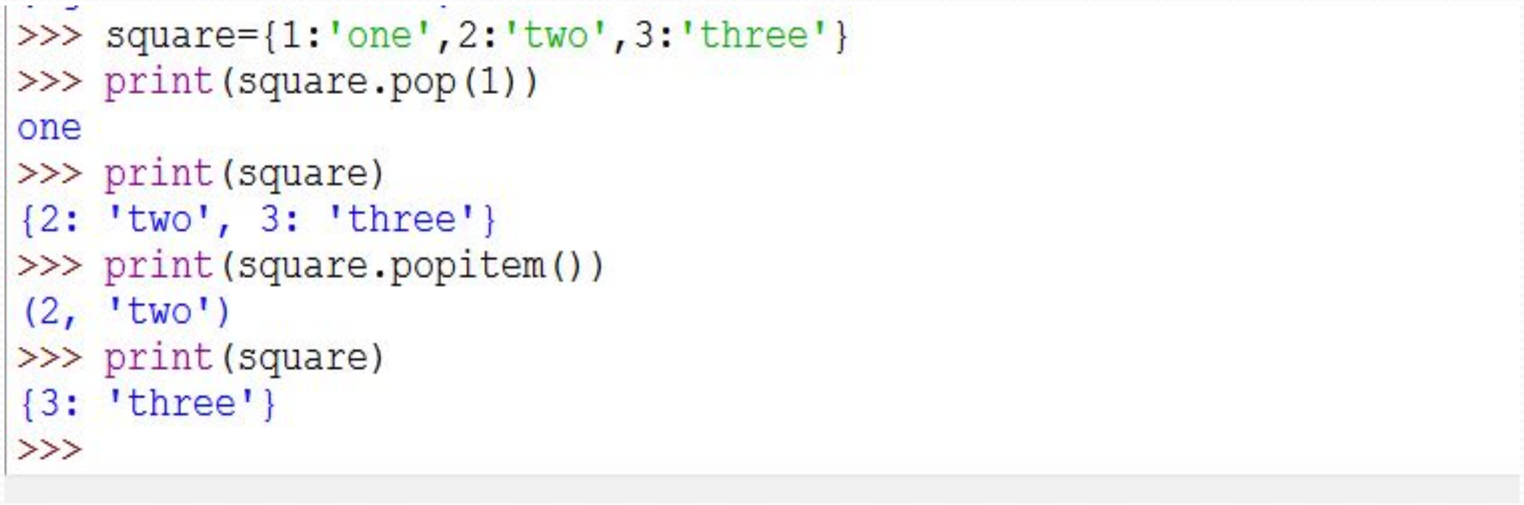


```
>>> my_dict={'name':'Honey','age':'23'}
>>> print(my_dict)
{'age': '23', 'name': 'Honey'}
>>> my_dict['gender']='Female'
>>> print(my_dict)
{'age': '23', 'name': 'Honey', 'gender': 'Female'}
>>>
```

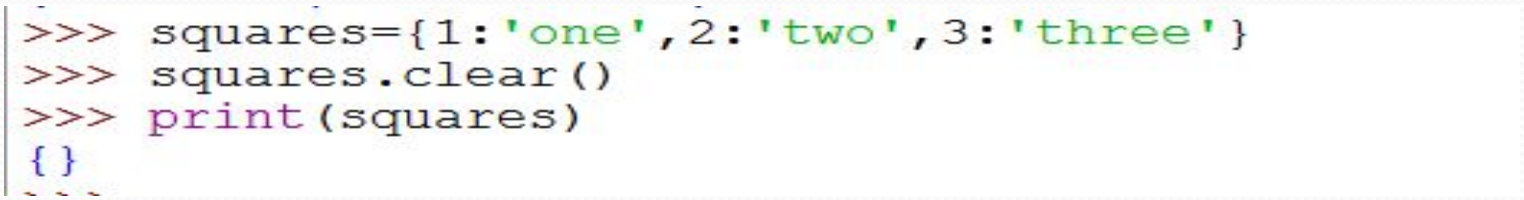
Removing Elements from Dictionary:

- **Pop():**We can remove a particular item in a dictionary by using this method.This method removes as item with the provided key and returns the value.
- **popitem():** can be used to remove and return an arbitrary item (key, value) form the dictionary.
- **Clear():** All the items can be removed at once using the clear () method.
- **del:** We can also use the del keyword to remove individual items or the entire dictionary itself.

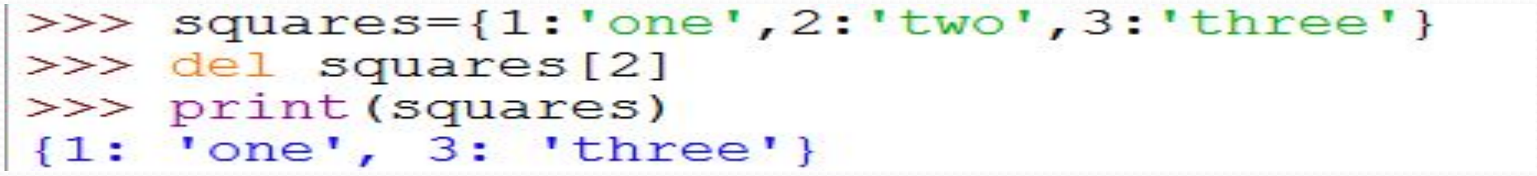
Example



```
>>> square={1:'one',2:'two',3:'three'}
>>> print(square.pop(1))
one
>>> print(square)
{2: 'two', 3: 'three'}
>>> print(square.popitem())
(2, 'two')
>>> print(square)
{3: 'three'}
>>>
```

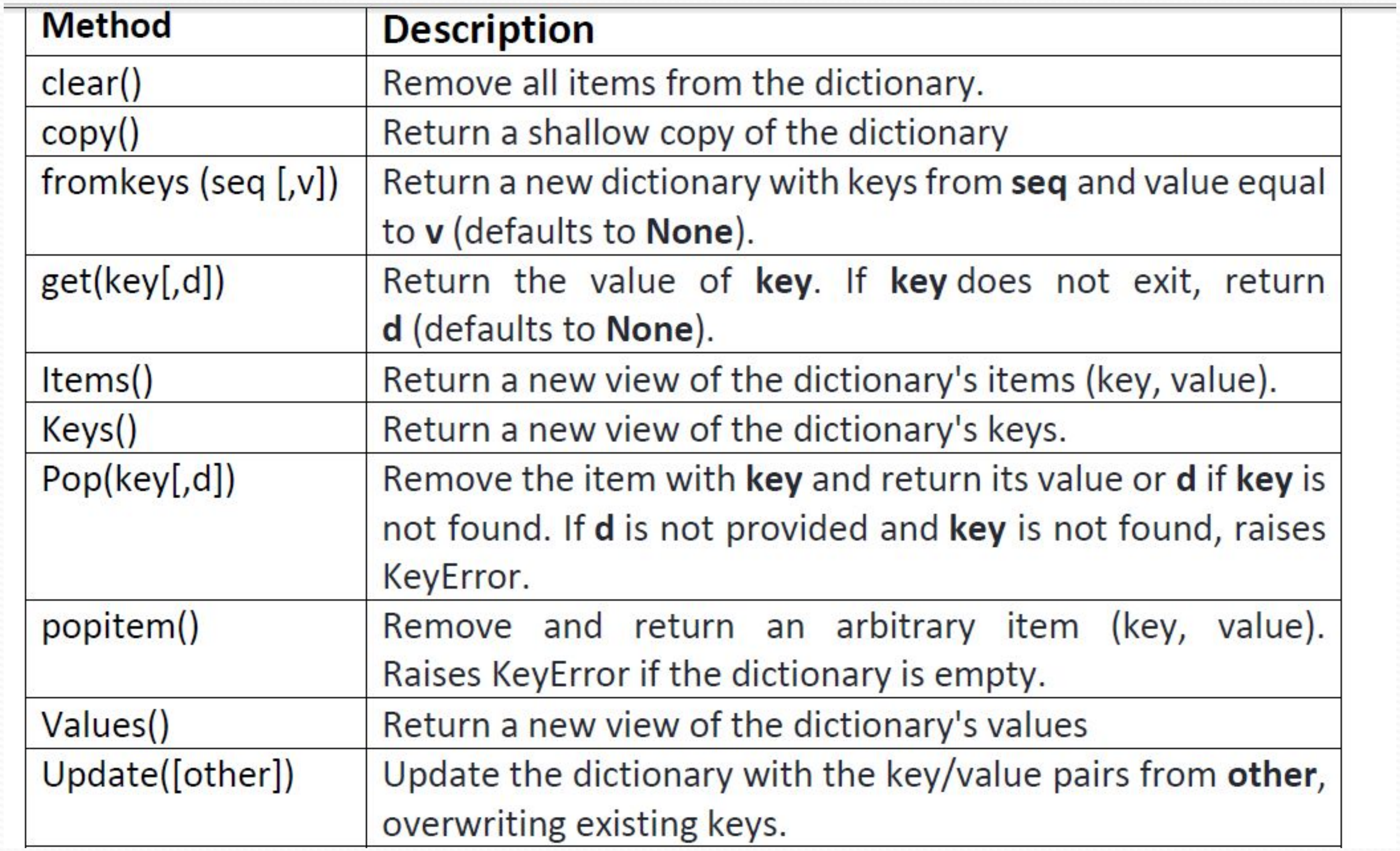


```
>>> squares={1:'one',2:'two',3:'three'}
>>> squares.clear()
>>> print(squares)
{}
```



```
>>> squares={1:'one',2:'two',3:'three'}
>>> del squares[2]
>>> print(squares)
{1: 'one', 3: 'three'}
```

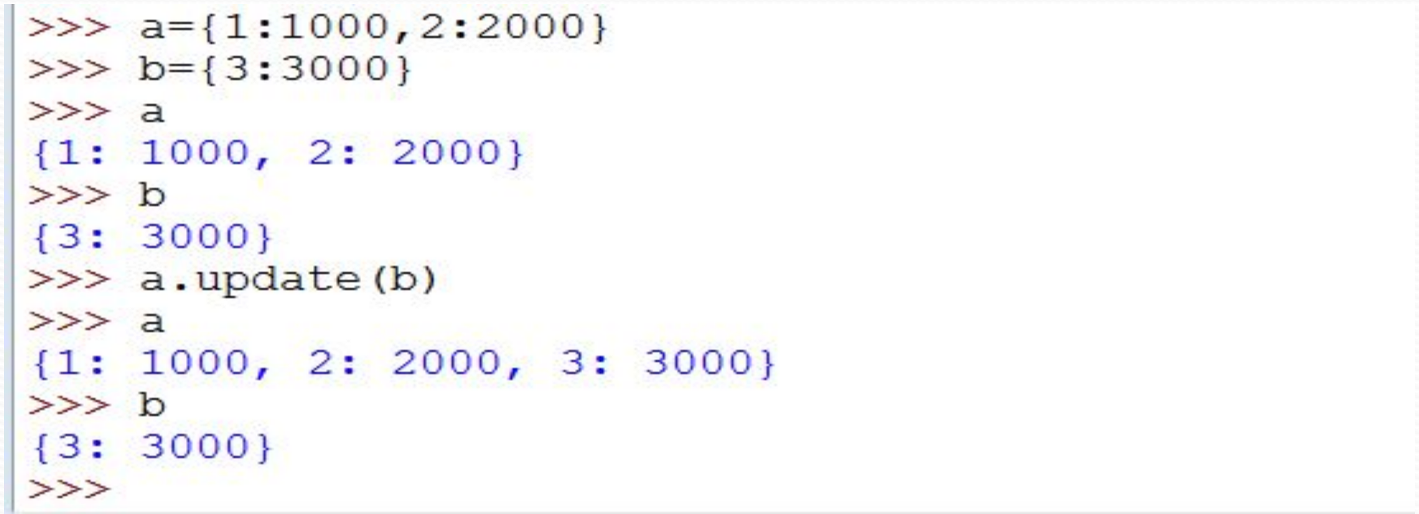
Methods of Dictionary



Method	**Description**
clear()	Remove all items from the dictionary.
copy()	Return a shallow copy of the dictionary
fromkeys (seq [,v])	Return a new dictionary with keys from **seq** and value equal to **v** (defaults to **None**).
get(key[,d])	Return the value of **key**. If **key** does not exit, return **d** (defaults to **None**).
Items()	Return a new view of the dictionary's items (key, value).
Keys()	Return a new view of the dictionary's keys.
Pop(key[,d])	Remove the item with **key** and return its value or **d** if **key** is not found. If **d** is not provided and **key** is not found, raises KeyError.
popitem()	Remove and return an arbitrary item (key, value). Raises KeyError if the dictionary is empty.
Values()	Return a new view of the dictionary's values
Update([other])	Update the dictionary with the key/value pairs from **other**, overwriting existing keys.

Update

- **The update**() method updates the dictionary with the elements from the another dictionary object or from an iterable of key/value pairs.
- The update() method adds element(s) to the dictionary if the key is not in the dictionary. If the key is in the dictionary, it updates the key with the new value.
- **<u>syntax of update() is</u>**:
- dict.update([other])



```
>>> a={1:1000,2:2000}
>>> b={3:3000}
>>> a
{1: 1000, 2: 2000}
>>> b
{3: 3000}
>>> a.update(b)
>>> a
{1: 1000, 2: 2000, 3: 3000}
>>> b
{3: 3000}
>>>
```

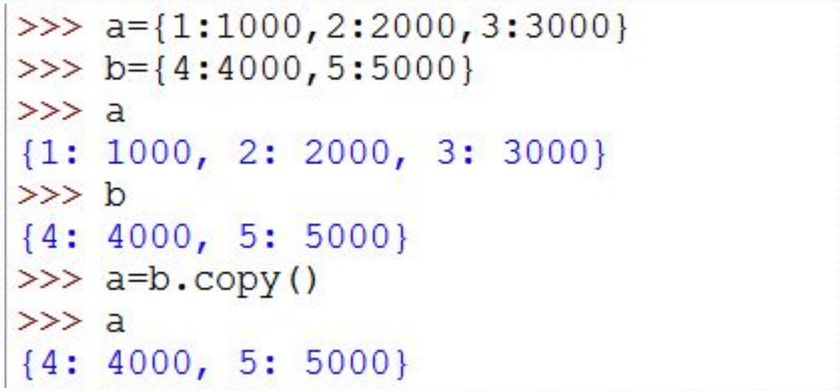
Copy Dictionaries

- There are Two ways to Copy Dictionaries
- **1. Copy()**
- **2. = Operator**

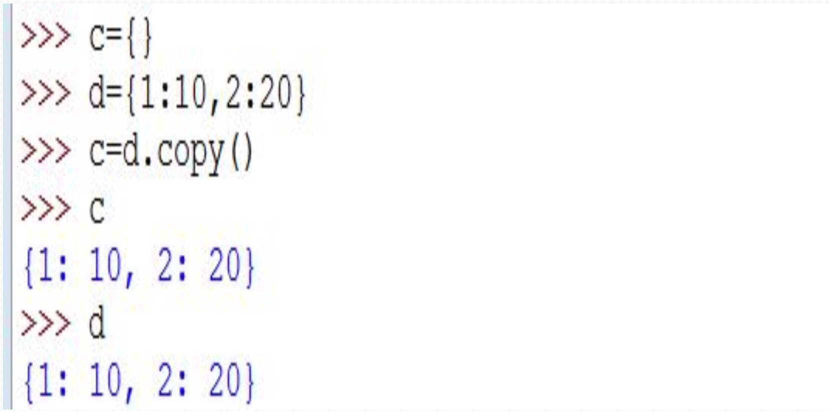
Continue….

They copy() method returns a shallow copy of the dictionary

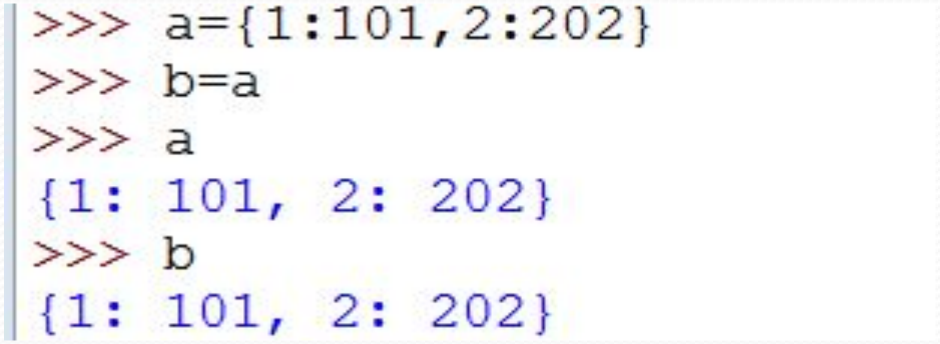
Syntax of copy() is: **dict.copy()**



```
>>> a={1:1000,2:2000,3:3000}
>>> b={4:4000,5:5000}
>>> a
{1: 1000, 2: 2000, 3: 3000}
>>> b
{4: 4000, 5: 5000}
>>> a=b.copy()
>>> a
{4: 4000, 5: 5000}
```



```
>>> c={}
>>> d={1:10,2:20}
>>> c=d.copy()
>>> c
{1: 10, 2: 20}
>>> d
{1: 10, 2: 20}
```



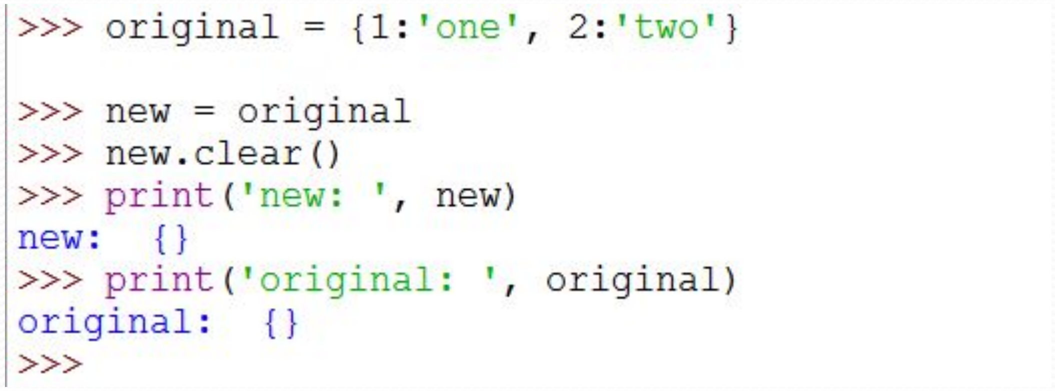
```
>>> a={1:101,2:202}
>>> b=a
>>> a
{1: 101, 2: 202}
>>> b
{1: 101, 2: 202}
```

Difference between copy() and = operator

- When copy() method is used, a new dictionary is created which is filled with a copy of the references from the original dictionary.
- When = operator is used, a new reference to the original dictionary is created.

Example

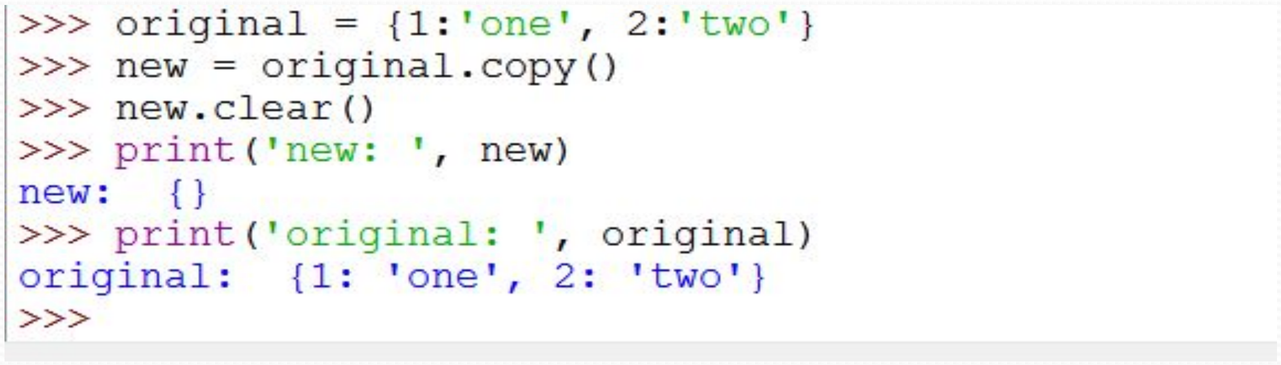
Using = operator



```
>>> original = {1:'one', 2:'two'}

>>> new = original
>>> new.clear()
>>> print('new: ', new)
new:  {}
>>> print('original: ', original)
original:  {}
>>>
```

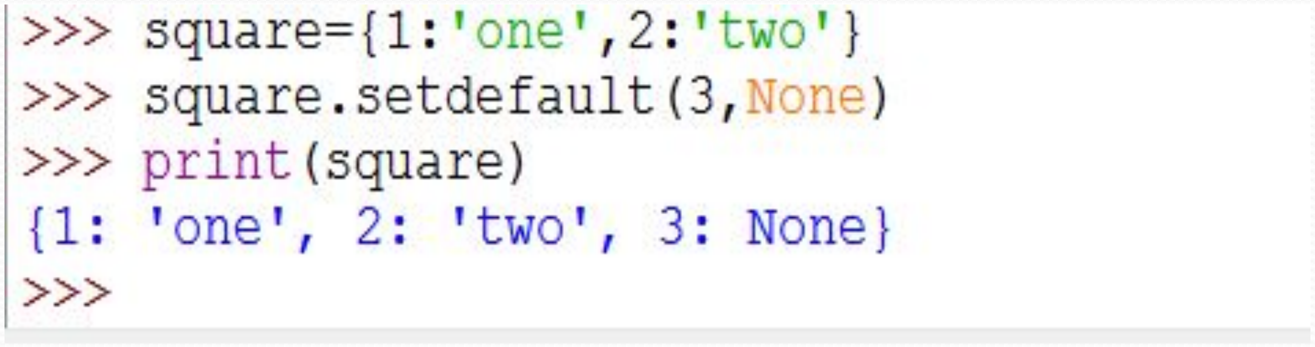
Using copy()



```
>>> original = {1:'one', 2:'two'}
>>> new = original.copy()
>>> new.clear()
>>> print('new: ', new)
new:  {}
>>> print('original: ', original)
original:  {1: 'one', 2: 'two'}
>>>
```

Set-default

- **Setdefault()** is similar to get(), but will set dict[key]=default , if key is not already in dict.
- **Syntax**
- dict.setdefault(key, default=None)

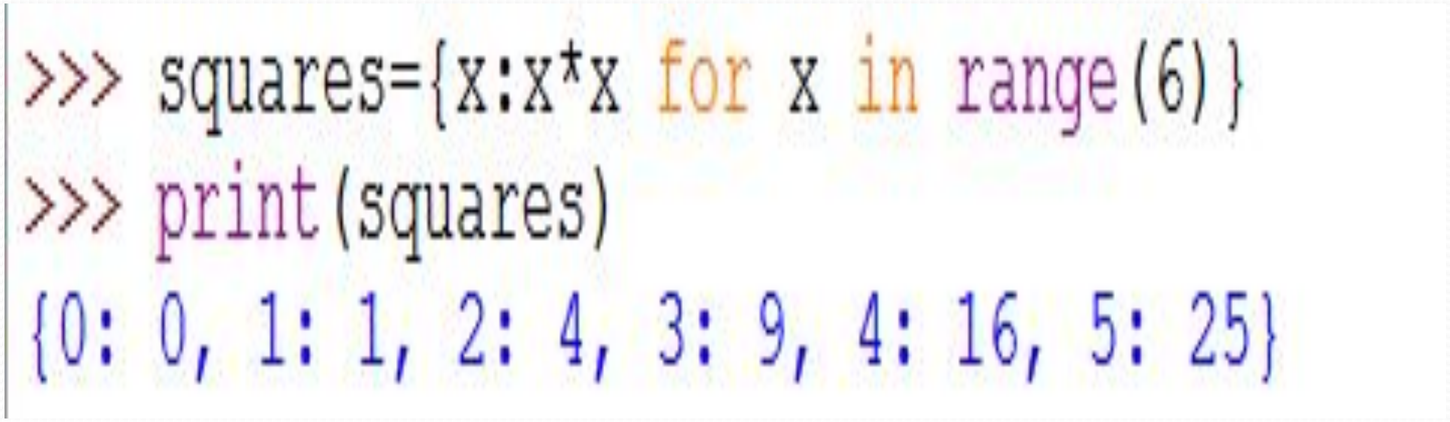


```
>>> square={1:'one',2:'two'}
>>> square.setdefault(3,None)
>>> print(square)
{1: 'one', 2: 'two', 3: None}
>>>
```

Dictionary Comprehension:

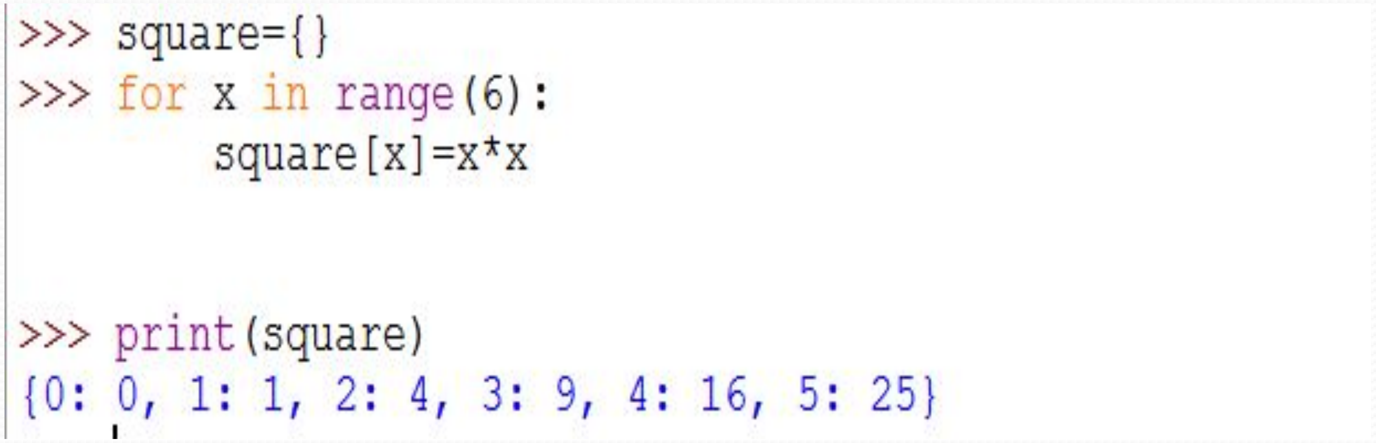
- Dictionary comprehension is an elegant and concise way to create new dictionary from an iterable in Python.
- Dictionary comprehension consists of an expression pair (key: value) followed by **for statement inside curly braces {}.**

Continue.....



```
>>> squares={x:x*x for x in range(6)}
>>> print(squares)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

This code is equivalent to

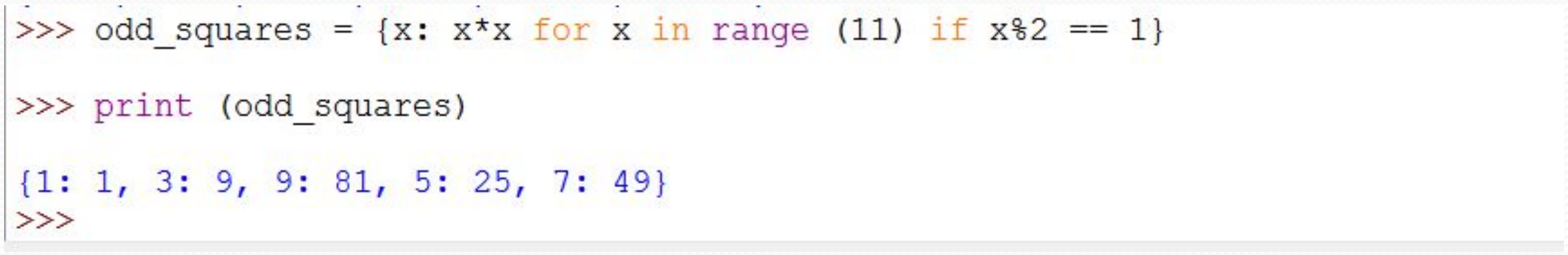


```
>>> square={}
>>> for x in range(6):
        square[x]=x*x


>>> print(square)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Continue….

- A dictionary comprehension can optionally contain more for or if Statements. An optional if statement can filter out items to form the new dictionary. Here are some examples to make dictionary with only odd items.



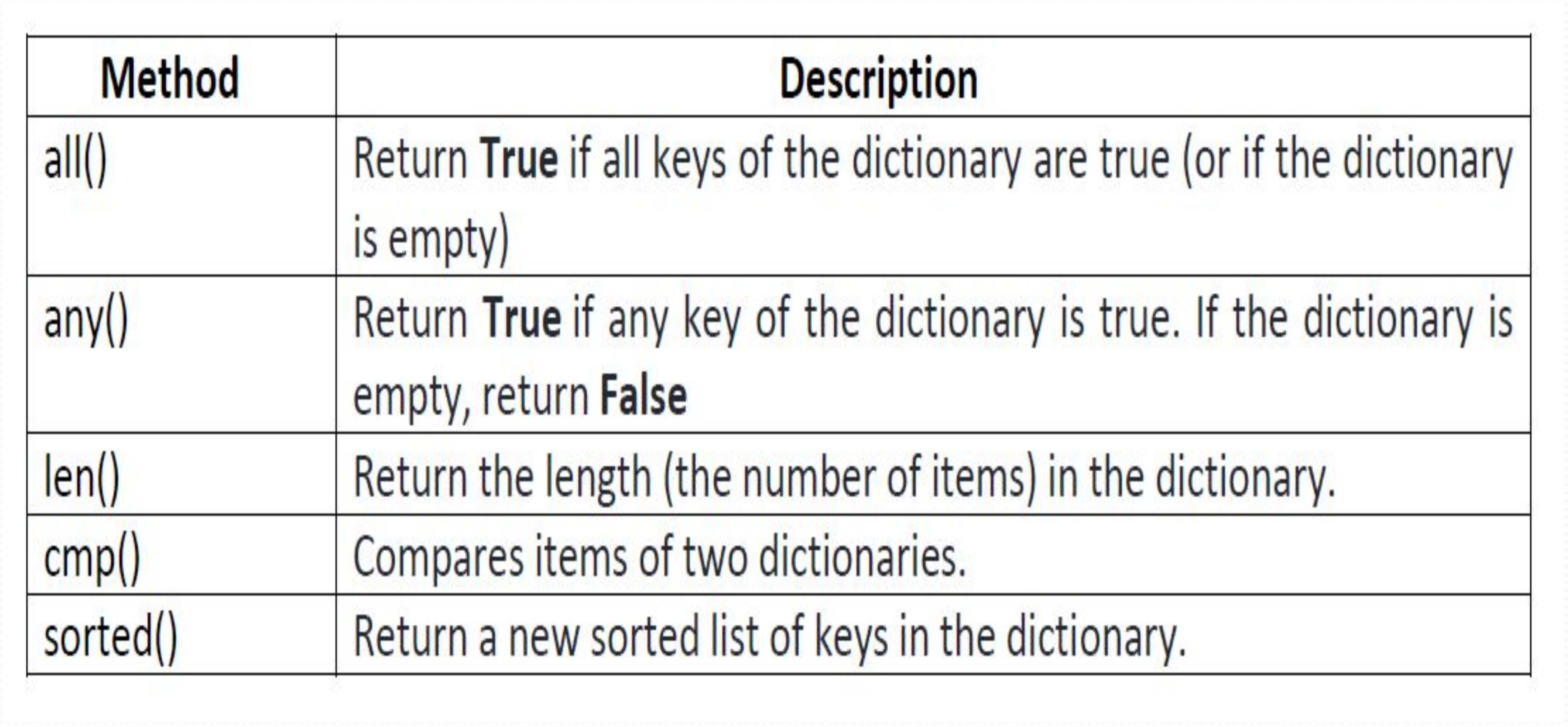
```
>>> odd_squares = {x: x*x for x in range (11) if x%2 == 1}

>>> print (odd_squares)

{1: 1, 3: 9, 9: 81, 5: 25, 7: 49}
>>>
```

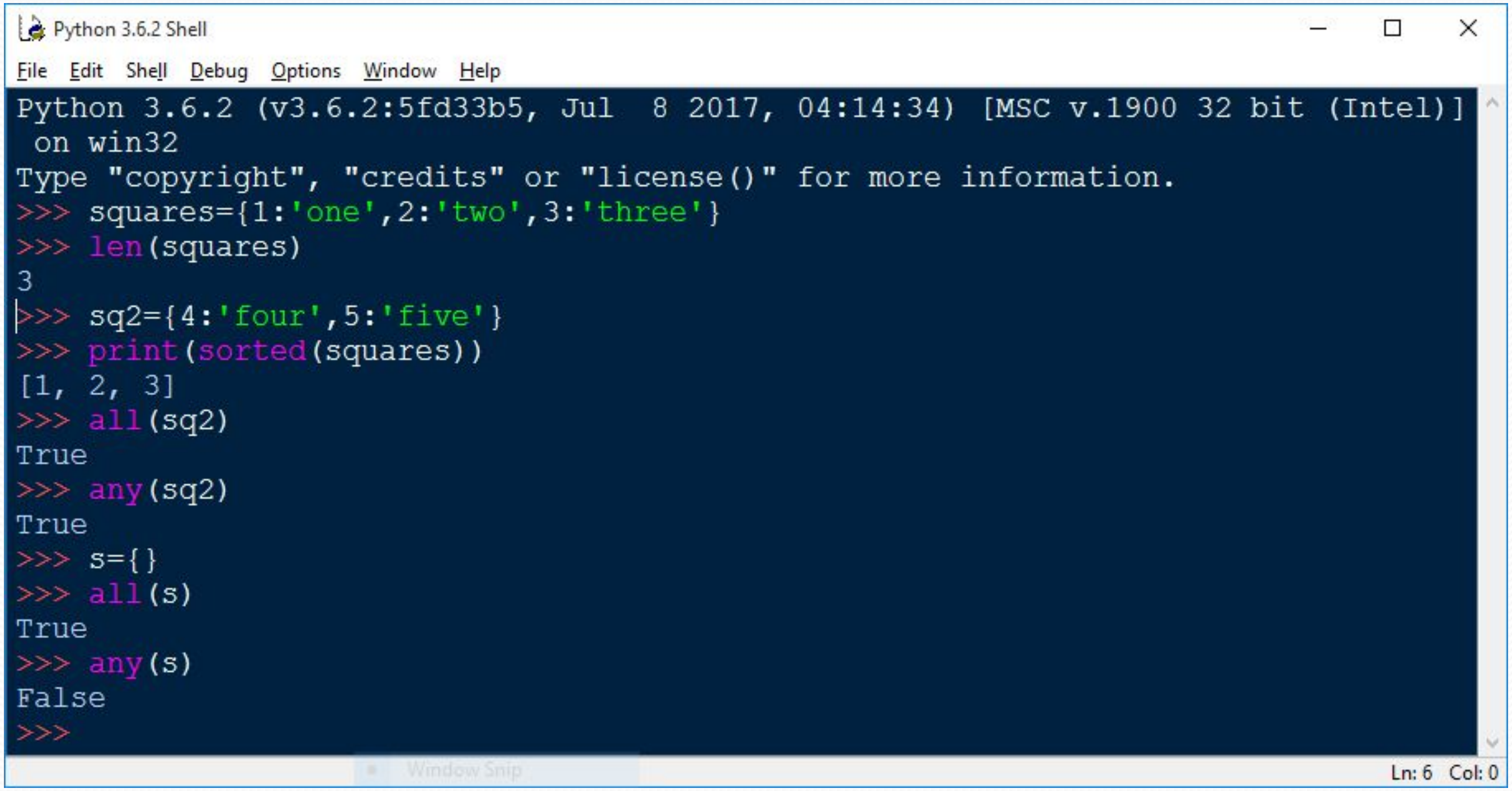
Built-In Functions

Built-in functions like all (), any (), len (), cmp (), sorted () etc. are commonly used with dictionary to perform different tasks



Method	Description
all()	Return **True** if all keys of the dictionary are true (or if the dictionary is empty)
any()	Return **True** if any key of the dictionary is true. If the dictionary is empty, return **False**
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries.
sorted()	Return a new sorted list of keys in the dictionary.

Continue….



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> squares={1:'one',2:'two',3:'three'}
>>> len(squares)
3
>>> sq2={4:'four',5:'five'}
>>> print(sorted(squares))
[1, 2, 3]
>>> all(sq2)
True
>>> any(sq2)
True
>>> s={}
>>> all(s)
True
>>> any(s)
False
>>>
```