Python Dictionaries Reference

Operator `[ ]`

Usage

```
dictionary[key]
```

Description

Return the value associated with `key`.

If `key` is not present in the dictionary, a `KeyError` exception is raised.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Access existing key
print(d['John'])         # Prints 20

# Access non-existing key
print(d['Kevin'])        # Raises KeyError exception
```

Built-in function `all()`

Usage

```
all(dictionary)
```

Description

The `all` built-in function accepts a dictionary as an argument and returns a Boolean value indicating whether all keys in *`dictionary`* are `True`. If any key in the dictionary is not a Boolean value, it is first converted using the same rules as the `bool()` type conversion function uses, and without affecting the state of the dictionary.

If the dictionary is empty, the `all()` function returns `True`.

Example

```
print(all({ }))                             # Prints True
print(all({ False: '', True: '' }))         # Prints False
print(all({ True: '', 1: '' }))             # Prints True
print(all({ True: '', 0: '' }))             # Prints False
```

Built-in function `any()`

Usage

```
any(dictionary)
```

Description

The `any` built-in function accepts a list as a dictionary and returns a Boolean value indicating whether at least one of the keys in *`dictionary`* is `True`. If any key in the dictionary is not a Boolean value, it is first converted using the same rules as the `bool()` type conversion function uses, and without affecting the state of the dictionary.

If the dictionary is empty, the `any()` function returns `False`.

Example

```
print(any({ }))                          # Prints False
print(any({ False: '' }))                # Prints False
print(any({ False: '', True: '' }))      # Prints True
print(any({ False: '', 0: '' }))         # Prints False
print(any({ False: '', 1: '' }))         # Prints True
```

Built-in function `len()`

Usage

```
len(dictionary)
```

Description

The `len()` built-in function returns the number of key-value pairs present in *`dictionary`*.

Example

```
d = { 'John': 20, 'Mary': 25 }
print(len(d))                       # Prints 2
```

Built-in function `max()`

Usage

```
max(dictionary)
```

Description

The `max()` built-in function returns the maximum key among all keys present in *`dictionary`*. The values in the key-value pairs are disregarded by this operation.

Example

```
d = {
  'John': 20,
  'Zack': 22,
  'Mary': 25
}
print(max(d))        # Prints 'Zack'
```

Built-in function `min()`

Usage

```
```

Description

The `min()` built-in function returns the minimum key among all keys present in *`dictionary`*. The values in the key-value pairs are disregarded by this operation.

Example

```
d = {
  2094: 'C++',
  1012: 'Python',
  2301: 'Javascript'
}
print(min(d))        # Prints 1012
```

Built-in function `sorted()`

Usage

```
sorted(dictionary)
```

Description

Built-in function `sorted()` accepts a dictionary as an argument, and returns a list containing all keys in the dictionary sorted in ascending order.

Example

```
d = {
  'John': 20,
  'Zack': 22,
  'Mary': 25
}
print(sorted(d))   # Prints ['John', 'Mary', 'Zack']
```

Built-in function `sum()`

Usage

```
sum(dictionary)
```

Description

Obtain the sum of all keys in the dictionary. If the dictionary is empty, this function returns 0.

Example

```
d = {
  1: 'a',
  2: 'b',
  3: 'c'
}
print(sum(d))     # Prints 6
print(sum({ }))   # Prints 0
```

Method `clear()`

Usage

```
dictionary.clear()
```

Description

Clear the content of the given dictionary and turn it into an empty dictionary.

Example

```
d = {
  'John': 20,
  'Zack': 22,
  'Mary': 25
}
d.clear()
print(d)   # Prints { }
```

Method `copy()`

Usage

```
dictionary.copy()
```

Description

Return a new dictionary containing all elements present in *`dictionary`*.

Example

```
d1 = { 'John': 10, 'Mary': 20, 'Kevin': 30 }
d2 = d1             # Variable 'd2' points to the same dictionary
d2['John'] = 40     # Changes in 'd2' affect 'd1'
print(d1)           # Prints { 'John': 40, 'Mary': 20, 'Kevin': 30 }
print(d2)           # Prints { 'John': 40, 'Mary': 20, 'Kevin': 30 }

d1 = { 'John': 10, 'Mary': 20, 'Kevin': 30 }
d2 = d1.copy()      # Variable 'd2' points to a new dictionary
d2['John'] = 40     # Changes in 'd2' are separate
print(d1)           # Prints { 'John': 10, 'Mary': 20, 'Kevin': 30 }
print(d2)           # Prints { 'John': 40, 'Mary': 20, 'Kevin': 30 }
```

Method `fromkeys()`

Usage

```
dictionary.fromkeys(sequence, value = None)
```

Description

Create a new dictionary, where each of its keys is obtained from argument *`sequence`*. Each key is mapped to the given value, or to `None` if argument *`value`* is not given. The dictionary that this method is applied on does not affect the behavior of the method.

Example

```
d = { }

vowels = d.fromkeys(['a', 'e', 'i', 'o', 'u'])
print(vowels)   # Prints {'a': None, 'e': None, 'i': None, 'o': None, 'u': None}

vowels = d.fromkeys(['a', 'e', 'i', 'o', 'u'], 0)
print(vowels)   # Prints {'a': 0, 'e': 0, 'i': 0, 'o': 0, 'u': 0}
```

Method `get()`

Usage

```
dictionary.get(key, default = None)
```

Description

Return the value for the given key in the dictionary. If the key is not present in the dictionary, this function returns the value given in argument default. If the key is not present and argument `default` is not given, the function returns `None`.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Access existing key
print(d.get('John'))         # Prints 20

# Access non-existing key with no default value
print(d.get('Kevin'))        # Prints None

# Access non-existing key with default value
print(d.get('Kevin', 30))    # Prints 30
```

Method `has_key()`

Usage

```
dictionary.has_key(key)
```

Description

Return whether the given key is present in the dictionary.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Check for presence of keys
print(d.has_key('John'))     # Prints True
print(d.has_key('Kevin'))    # Prints False
```

Method `items()`

Usage

```
dictionary.items()
```

Description

Return a list of tuples representing all key-value pairs present in the dictionary. The first element of each tuple represents each key, and the second element represents the value mapped to it.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Obtain list of tuples
print(d.items())       # Prints [ ('John', 20), ('Mary', 25) ]
```

Method `keys()`

Usage

```
dictionary.keys()
```

Description

Return a list with all keys present in the dictionary.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Obtain list of keys
print(d.keys())       # Prints [ 'John', 'Mary' ]
```

Method `pop()`

Usage

```
dictionary.pop(key, [default])
```

Description

Remove the given key from the dictionary and return its associated value. If *`key`* is not present in the dictionary and argument *`default`* is given, the function returns the value in argument *`default`*. If the key is not present in the dictionary and argument *`default`* is not given, the function raises a `KeyError` exception.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Remove key 'John'
print(d.pop('John'))         # Prints 20
print(d)                     # Prints { 'Mary': 25 }

# Try to remove non-existing key with a default value
print(d.pop('Kevin', 30))    # Prints 30

# Try to remove non-existing key with no default value
print(d.pop('Kevin'))        # Raises 'KeyError' exception
```

Method `popitem()`

Usage

```
dictionary.popitem()
```

Description

Remove the last key inserted in the dictionary and return a tuple of the form `(key, value)`. If the dictionary is empty, a `KeyError` exception is raised.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Remove last key-value pair
print(d.popitem())          # Prints ('Mary', 25)
print(d)                    # Prints { 'John': 20 }

# Remove last key-value pair again
print(d.popitem())          # Prints ('John', 20)
print(d)                    # Prints { }

# Attempt to remote last key-value pair on an empty dictionary
d.popitem()                 # Raises 'KeyError' exception
```

Method `update()`

Usage

```
dictionary1.update(dictionary2)
```

Description

Updates *`dictionary1`* with the key-value pairs present in *`dictionary2`*, overwriting existing keys.

Example

```
# Create dictionaries
d1 = { ‘John’: 20, ‘Mary’: 30 }
d2 = { ‘Mary’: 40, ‘Susan’: 50 }

# Update 'd1' with keys in 'd2'
d1.update(d2)
print(d1)          # Prints {‘John’: 20, ‘Mary’: 40, ‘Susan’: 50}
print(d2)          # Prints {‘Mary’: 40, ‘Susan’: 50}
```

Method `values()`

Usage

```
dictionary.keys()
```

Description

Return a list with all values present in the dictionary.

Example

```
# Create dictionary
d = { 'John': 20, 'Mary': 25 }

# Obtain list of values
print(d.values())       # Prints [ 20, 25 ]
```