Dictionary

Python dictionary is an unordered collection of items. While other compound datatypes have only value as an element, a dictionary has a key: value pair. Dictionaries are optimized to retrieve values when the key is known.

Ceating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value. While values can be of any datatype and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique. We can also create a dictionary using the built-in function dict().

# empty dictionary

my\_dict = {}

# dictionary with integer keys

my\_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys

my\_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()

my\_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair

my\_dict = dict([(1,'apple'), (2,'ball')])

Accessing Elements in a Dictionary

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method. The difference while using get() is that it returns None instead of KeyError, if the key is not found.

>>> my\_dict = {'name':'Ranjit', 'age': 26}

>>> my\_dict['name']

'Ranjit'

>>> my\_dict.get('age')

26

>>> my\_dict.get('address')

>>> my\_dict['address']

...

KeyError: 'address'

Changing or Adding Elements in a 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

{'age': 26, 'name': 'Ranjit'}

>>> my\_dict['age'] = 27 # update value

>>> my\_dict

{'age': 27, 'name': 'Ranjit'}

>>> my\_dict['address'] = 'Downtown' # add item

>>> my\_dict

{'address': 'Downtown', 'age': 27, 'name': 'Ranjit'}

Deleting or Removing Elements from a Dictionary

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

>>> squares = {1:1, 2:4, 3:9, 4:16, 5:25} # create a dictionary

>>> squares.pop(4) # remove a particular item

16

>>> squares

{1: 1, 2: 4, 3: 9, 5: 25}

>>> squares.popitem() # remove an arbitrary item

(1, 1)

>>> squares

{2: 4, 3: 9, 5: 25}

>>> del squares[5] # delete a particular item

>>> squares

{2: 4, 3: 9}

>>> squares.clear() # remove all items

>>> squares

{}

>>> del squares # delete the dictionary itself

>>> squares

...

NameError: name 'squares' is not defined

Python Dictionary Methods

Methods that are available with dictionary are tabulated below. Some of them have already been used in the above examples.

|  |  |
| --- | --- |
| Python Dictionary Methods | |
| Method | Description |
| clear() | Remove all items form 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 keydoesnot exit, return d (defaults toNone). |
| 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 andkey is not found, raisesKeyError. |
| popitem() | Remove and return an arbitary item (key, value). Raises KeyErrorif the dictionary is empty. |
| setdefault(key[,d]) | If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None). |
| update([other]) | Update the dictionary with the key/value pairs from other, overwriting existing keys. |
| values() | Return a new view of the dictionary's values |

Here are a few example use of these methods.

>>> marks = {}.fromkeys(['Math','English','Science'], 0)

>>> marks

{'English': 0, 'Math': 0, 'Science': 0}

>>> for item in marks.items():

... print(item)

...

('English', 0)

('Math', 0)

('Science', 0)

>>> list(sorted(marks.keys()))

['English', 'Math', 'Science']

Python 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 {}. Here is an example to make a dictionary with each item being a pair of a number and its square.

>>> squares = {x: x\*x for x in range(6)}

>>> squares

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

This code is equivalent to

squares = {}

for x in range(6):

squares[x] = x\*x

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}

>>> odd\_squares

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

Other Dictionary Operations

Dictionary Membership Test

We can test if a key is in a dictionary or not using the keyword in. Notice that membership test is for keys only, not for values.

>>> squares

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

>>> 1 in squares

True

>>> 2 not in squares

True

>>> # membership tests for key only not value

>>> 49 in squares

False

Iterating Through a Dictionary

Using a for loop we can iterate though each key in a dictionary.

>>> squares

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

>>> for i in squares:

... print(squares[i])

...

1

9

81

25

49

Dictionary comprehension

**>>> D = {x: x\*\*2 for x in [1,2,3,4,5]}**

**>>> D**

**{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}**

**>>> D = {x.upper(): x\*3 for x in 'abcd'}**

**>>> D**

**{'A': 'aaa', 'C': 'ccc', 'B': 'bbb', 'D': 'ddd'}**

When we want initialize a dict from keys, we do this:

**>>> D = dict.fromkeys(['a','b','c'], 0)**

**>>> D**

**{'a': 0, 'c': 0, 'b': 0}**

>>> d = {n: n\*\*2 for n in range(5)}

>>> print d

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

>>> d = {n: True for n in range(5)}

>>> print d

{0: True, 1: True, 2: True, 3: True, 4: True}