**Dictionaries**

* Python's dictionaries are kind of hash table type which consist of **key-value** pairs

of **unordered** elements.

* + **Keys** : must be data types ,usually numbers or strings.
  + **Values** : can be any arbitrary Python object.
* Python Dictionaries are **mutable** objects that can change their values.
* A dictionary is enclosed by *curly braces* ({ }), the items are separated by *commas*, and each key is separated from its value by a *colon* (:).
* Dictionary’s values can be assigned and accessed using square braces ([]) with a key to obtain its value.
* Dictionary can also be created by the built-in function *dict()*. An empty dictionary can be created by just placing two curly braces {}.
* Dictionary keys are case sensitive, same name but different cases of *key* will be treated distinctly.

**Example:**

# Creating an empty Dictionary

a = {}

print("Empty Dictionary: ")

print(a)

# Creating a Dictionary with Integer Keys

a = {1: 'hello', 2: 'good', 3: 'morning'}

print("\nDictionary with the use of Integer Keys: ")

print(a)

# Creating a Dictionary with Mixed keys

a = {'Name': 'Wozti', 1: [1, 2, 3, 4]}

print("\nDictionary with the use of Mixed Keys: ")

print(a)

# Creating a Dictionary with dict() method

a = dict({1: 'hello', 2: 'good', 3:'morning'})

print("\nDictionary with the use of dict(): ")

print(a)

# Creating a Dictionary with each item as a Pair

Dict = dict( [ (1, 'Wozti'), (2, 'Academy') ] )

print("\nDictionary with each item as a pair: ")

print(Dict)

**Result:**

Empty Dictionary:

{}

Dictionary with the use of Integer Keys:

{1: 'hello', 2: 'good', 3: 'morning'}

Dictionary with the use of Mixed Keys:

{'Name': 'Wozti', 1: [1, 2, 3, 4]}

Dictionary with the use of dict():

{1: 'hello', 2: 'good', 3: 'morning'}

Dictionary with each item as a pair:

{1: 'Wozti', 2: 'Academy'}

**Example:**

# Creating a Nested Dictionary

a = {1: 'Hello', 2: 'Pupils', 3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Wozti'}}

print(a)

**Result:**

{1: 'Hello', 2: 'Pupils', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Wozti'}}

**Example:**

# Adding more values to a single Key

a = dict({1: 'hello', 2: 'good', 3:'morning'})

a['hai'] = 2, 3, 4

print("\nDictionary after adding 3 elements: ")

print(a)

**Result:**

Dictionary after adding 3 elements:

{1: 'hello', 2: 'good', 3: 'morning', 'hai': (2, 3, 4)}

**Example:**

a = dict({1: 'hello', 2: 'good', 3:'morning'})

# Adding nested Key-value to Dictionary

a[4] = {'1' : 'good', '2' : 'noon'}

print("\nAdding a Nested Key: ")

print(a)

**Result:**

Adding a Nested Key:

{1: 'hello', 2: 'good', 3: 'morning', 4: {'1': 'good', '2': 'noon'}}

**Example:**

a = dict({1: 'hello', 2: 'good', 3:'morning', 4: {'1' : 'good', '2' : 'noon'}})

print("\nDeleting a nested Key: ")

del a[4]['2']

print(a)

**Result:**

Deleting a nested Key:

{1: 'hello', 2: 'good', 3: 'morning', 4: {'1': 'good'}}

/////////////////////////////////////////

Updating an existing value in a Dictionary can be done by using the built-in *update()* method. Nested key-values can also be added to an existing Dictionary.

While adding a value, if the key-value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

/////////////////////////////////////////

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

Accessing Items

You can access the items of a dictionary by referring to its key name:

Example

Get the value of the "model" key:

x = thisdict["model"]

or

x = thisdict.get("model")

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018

* Print all key names in the dictionary, one by one:

for x in thisdict:  
  print(x)

* You can also use the values() function to return values of a dictionary:

for x in thisdict.values():  
  print(x)

* Loop through both keys and values, by using the items() function:

for x, y in thisdict.items():  
  print(x, y)

* To determine if a specific key is present in a dictionary use the ‘in’ keyword:

Check if "apple" is present in the tuple:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

[Run example »](https://www.w3schools.com/python/showpython.asp?filename=demo_dictionary_in)

Update()

a = {1,'hello'}

b = {2,'hai'}

a.update(b)

print(a)

result

{1, 2, 'hai', 'hello'}

////////////////////////////////////////////

Dictionary Length

To determine how many items (key-value pairs) a dictionary have, use the len() method.

Example

Print the number of items in the dictionary:

print(len(thisdict))

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)

Removing Items

There are several methods to remove items from a dictionary:

Example

The del keyword removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

* The pop() method removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

The popitem() method removes the last inserted item (in versions before 2.7, a random item is removed instead):

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.popitem()  
print(thisdict)

* The clear() keyword empties the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.clear()  
print(thisdict)

**Example:**

a = dict(one=1, two=2, three=3)

print(a)

b = {'one': 1, 'two': 2, 'three': 3}

print(b)

c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))

print(c)

d = dict([('two', 2), ('one', 1), ('three', 3)])

print(d)

e = dict({'three': 3, 'one': 1, 'two': 2})

print(e)

if (a == b == c == d == e):

print("Hello")

**Result:**

{'one': 1, 'two': 2, 'three': 3}

{'one': 1, 'two': 2, 'three': 3}

{'one': 1, 'two': 2, 'three': 3}

{'two': 2, 'one': 1, 'three': 3}

{'three': 3, 'one': 1, 'two': 2}

Hello

/////////////////////////////////////

In Python Dictionary, *setdefault()* method returns the value of a key (if the key is in dictionary). If not, it inserts key with a value to the dictionary.

**Syntax:**

dict.*setdefault*(key, default\_value)

**Parameters:**

 It takes two parameters:

*key* – Key to be searched in the dictionary.

*default\_value (optional)* –  Default\_value to be inserted to the dictionary if key is not in the dictionary. If not provided, the default\_value will be *None*.

///////////////////////////////

**Example:**

D = { 'A': 'good', 'B': 'bad', 'C': 'ugly'}

E = D.setdefault('C')

print("Dictionary:", D)

print("Third\_Value:", E)

**Result:**

Dictionary: {'A': 'good', 'B': 'bad', 'C': 'ugly'}

Third\_Value: ugly

/////////////////////////////

When key is not in the dictionary.

///////////////////////////

**Example:**

D = { 'A': 'good', 'B': 'bad'}

E = D.setdefault('C')

print("Dictionary:", D)

print("Third\_value:", E)

F = D.setdefault('D', 'ugly')

print("Dictionary:", D)

print("Fourth\_value:", F)

**Result:**

Dictionary: {'A': 'good', 'B': 'bad', 'C': None}

Third\_value: None

Dictionary: {'A': 'good', 'B': 'bad', 'C': None, 'D': 'ugly'}

Fourth\_value: ugly

**Example:**

d = {"one": 1, "two": 2, "three": 3, "four": 4}

print(list(reversed(d))) # Will produce error in python 3.7, whereas it may work in 3.8

print(list(reversed(d.values()))) # Will produce error in python 3.7, whereas it may work in 3.8

print(list(reversed(d.items()))) # Will produce error in python 3.7, whereas it may work in 3.8

**Result:**

['four', 'three', 'two', 'one']

[4, 3, 2, 1]

[('four', 4), ('three', 3), ('two', 2), ('one', 1)]

**Example:**

dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}

k = dishes.keys()

print(k)

print(type(k))

v = dishes.values()

print(v)

print(type(v))

n = 0

for val in v:

n += val

print(n)

print(list(k))

print(list(v))

print(k & {'eggs', 'bacon', 'salad'}) # set operation

print(k ^ {'sausage', 'juice'}) # set operation

print(k - {'sausage', 'juice'}) # set operation

print(k | {'sausage', 'juice'}) # set operation

#print(k.intersection(dishes)) # This set operation will produce error

**Result:**

dict\_keys(['eggs', 'sausage', 'bacon', 'spam'])

<class 'dict\_keys'>

dict\_values([2, 1, 1, 500])

<class 'dict\_values'>

504

['eggs', 'sausage', 'bacon', 'spam']

[2, 1, 1, 500]

{'bacon', 'eggs'}

{'eggs', 'spam', 'bacon', 'juice'}

{'spam', 'bacon', 'eggs'}

{'spam', 'sausage', 'eggs', 'bacon', 'juice'}

#### Dictionary Methods

|  |  |
| --- | --- |
| METHODS | DESCRIPTION |
| [copy()](https://www.geeksforgeeks.org/python-dictionary-copy/) | They *copy()* method returns a shallow copy of the dictionary. |
| [clear()](https://www.geeksforgeeks.org/python-dictionary-clear/) | The clear() method removes all items from the dictionary. |
| [pop()](https://www.geeksforgeeks.org/python-dictionary-pop-method/) | Removes and returns an element from a dictionary having the given key. |
| [popitem()](https://www.geeksforgeeks.org/python-dictionary-popitem-method/) | Removes the arbitrary key-value pair from the dictionary and returns it as tuple. |
| [get()](https://www.geeksforgeeks.org/get-method-dictionaries-python/) | It is a conventional method to access a value for a key. |
| [dictionary\_name.values()](https://www.geeksforgeeks.org/python-dictionary-values/) | returns a list of all the values available in a given dictionary. |
| str() | Produces a printable string representation of a dictionary. |
| [update()](https://www.geeksforgeeks.org/python-dictionary-update-method/) | Adds dictionary dict2’s key-values pairs to dict |
| [setdefault()](https://www.geeksforgeeks.org/python-dictionary-setdefault-method/) | Set dict[key]=default if key is not already in dict |
| [keys()](https://www.geeksforgeeks.org/python-dictionary-keys-method/) | Returns list of dictionary dict’s keys |
| [items()](https://www.geeksforgeeks.org/python-dictionary-items-method/) | Returns a list of dict’s (key, value) tuple pairs |
| [has\_key()](https://www.geeksforgeeks.org/python-dictionary-has_key/) | Returns *True* if key in dictionary, *False* otherwise.  ***Syntax:***  *dict.has\_key(key)*  ***key****– This is the Key to be searched in the dictionary.*  *has\_keys()* was removed in Python 3.x. We can use ‘in’ keyword instead. |
| [fromkeys()](https://www.geeksforgeeks.org/python-dictionary-fromkeys-method/) | The *fromkeys()* method returns a dictionary with the specified keys and values.  Syntax:  dict.fromkeys(keys, value)  *keys*:  Required. An iterable specifying the keys of the new dictionary  *value*:  Optional. The value for all keys. Default value is *None* |
| [type()](https://www.geeksforgeeks.org/python-type-function/) | Returns the type of the passed variable. |

**Example:**

x = ('key1', 'key2', 'key3')

y = 0

b = dict.fromkeys(x, y)

print(b)

**Result:**

{'key1': 0, 'key2': 0, 'key3': 0}

////////////////////////////////

Same example as above, but without specifying the value:

///////////////////////////////////

**Example:**

x = ('key1', 'key2', 'key3')

y = dict.fromkeys(x)

print(y)

**Result:**

{'key1': None, 'key2': None, 'key3': None}

**Example:**

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

b = thisdict.copy()

print(b)

**Result:**

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

**Example:**

from collections import OrderedDict

b = OrderedDict()

b['apple'] = 'fruit'

b['zoo'] = 'animal'

b['cat'] = 'pet'

b['lion'] = 'king'

b.move\_to\_end('zoo')

b.popitem(last=True)

print(b)

**Result:**

OrderedDict([('apple', 'fruit'), ('cat', 'pet'), ('lion', 'king')])

///////////////////////////////////////////////////////////////////////////////////////////

In Python, the ‘|’ operator can be used to merge two dictionaries together. The resulting dictionary will contain all the key-value pairs from both dictionaries. If a key is present in both dictionaries, the value from the second dictionary will overwrite the value from the first dictionary.

**Example**

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

merged\_dict = dict1 | dict2

print(merged\_dict)

**Result**

{'a': 1, 'b': 3, 'c': 4}

/////////////////////////////////////////////////////////////////

The ‘|=’ operator can be used to update one dictionary with the key-value pairs from another dictionary. This is similar to the ‘update()’ method, but it uses the ‘|’ operator to update the dictionary in-place.

**Example**

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

dict1 |= dict2

print(dict1)

**Result**

{'a': 1, 'b': 3, 'c': 4}

**Assignments**

1

Performing student details using dictionary.

2

Assignment on arranging dictionary items based on latest added item.

3

Assignment on identifying factors of list using dictionary.