

[Home](https://learning.ccbp.in/)Code Playground

Nakka Vijay Kumar

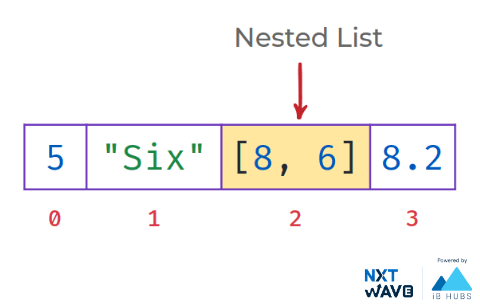
**NK**

**Cheat Sheet**

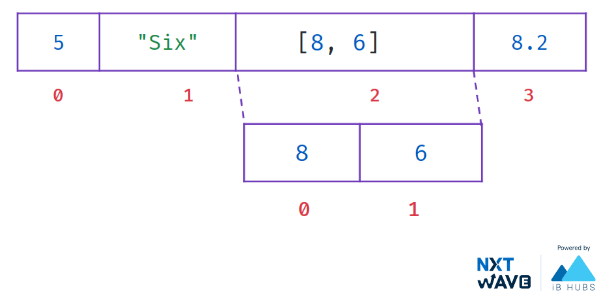
Cheat Sheet

**Nested Lists & String Formatting**

A list as an item of another list.



Accessing Nested List



**Code**



1

2

list\_a = [5, "Six", [8, 6], 8.2]

print(list\_a[2])

PYTHON

**Output**



[8, 6]

Accessing Items of Nested List

*Example - 1*

**Code**



1

2

list\_a = [5, "Six", [8, 6], 8.2]

print(list\_a[2][0])

PYTHON

**Output**



8

*Example - 2*

**Code**



1

2

list\_a = ["Five", "Six"]

print(list\_a[0][1])

PYTHON

**Output**



i

**String Formatting**

**Code**



1

2

3

4

name = input()

age = int(input())

msg = ("Hi " + name + ". You are "+ str(age) + " years old.")

print(s)

PYTHON

String formatting simplifies this concatenation.

It increases the readability of code and type conversion is not required.

Add Placeholders

Add placeholders

{}

where the string needs to be formatted.



1

2

msg = "Hi {}. You are {} years old."

msg.format(val\_1, val\_2,..)

PYTHON

Inserts values inside the string’s placeholder

{}

**Code**



1

2

3

4

name = "Raju"

age = 10

msg = "Hi {}. You are {} years old."

print(msg.format(name, age))

PYTHON

**Output**



Hi Raju. You are 10 years old.

Number of Placeholders

**Code**



1

2

3

4

name = "Raju"

age = 10

msg = "Hi {}. You are {} years old {}."

print(msg.format(name, age))

PYTHON

**Output**



IndexError: Replacement index 2 out of range for positional args tuple

Numbering Placeholders

Numbering placeholders, will fill values according to the position of arguments.

**Code**



1

2

3

4

name = input()

age = int(input())

msg = "Hi {0}. You are {1} years old."

print(msg.format(name, age))

PYTHON

**Output**



Hi Raju. You are 10 years old.

**Code**



1

2

3

4

name = input()

age = int(input())

msg = "Hi {1}. You are {0} years old."

print(msg.format(name, age))

PYTHON

**Output**



Hi 10. You are Raju years old.

Naming Placeholder

Naming placeholders will fill values according to the keyword arguments.

**Code**



1

2

3

4

name = input()

age = int(input())

msg = "Hi {name}. You are {age} years old."

print(msg.format(name=name, age=age))

PYTHON

**Output**



Hi Raju. You are 10 years old.

Notes

Discussions

Notes

NEW NOTE

CANCELSAVE

[**Download App**](https://play.google.com/store/apps/details?id=co.ibhubs.chat)

**Personal Manager**





[Home](https://learning.ccbp.in/)Code Playground

Nakka Vijay Kumar

**NK**

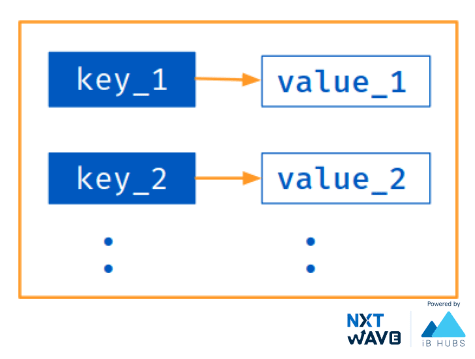
**Cheat Sheet**

Cheat Sheet

# Dictionaries

Unordered collection of items.

Every dictionary item is a **Key-value** pair.



## Creating a Dictionary

Created by enclosing items within **{curly}** brackets

Each item in dictionary has a key - value pair separated by a **comma**.

#### Code



1

2

3

4

dict\_a = {

"name": "Teja",

"age": 15

}

PYTHON

### Key - Value Paris

#### Code



1

2

3

4

dict\_a = {

"name": "Teja",

"age": 15

}

PYTHON

In the above dictionary, the

* keys are

name

 and

age

* values are

Teja

 and

15

### Collection of Key-Value Pairs

#### Code



1

2

3

4

dict\_a = { "name": "Teja",

"age": 15 }

print(type(dict\_a))

print(dict\_a)

PYTHON

#### Output



<class 'dict'>

{'name': 'Teja','age': 15}

## Immutable Keys

Keys must be of immutable type and must be unique.

Values can be of any data type and can repeat.

#### Code



1

2

3

4

5

dict\_a = {

"name": "Teja",

"age": 15,

"roll\_no": 15

}

PYTHON

## Creating Empty Dictionary

#### Code - 1



1

2

3

dict\_a = dict()

print(type(dict\_a))

print(dict\_a)

PYTHON

#### Output



<class 'dict'>

{}

#### Code - 2



1

2

3

dict\_a = {}

print(type(dict\_a))

print(dict\_a)

PYTHON

#### Output



<class 'dict'>

{}

## Accessing Items

To access the items in dictionary, we use square bracket

[ ]

along with the

key

to obtain its value.

#### Code



1

2

3

4

5

dict\_a = {

'name': 'Teja',

'age': 15

}

print(dict\_a['name'])

PYTHON

#### Output



Teja

## Accessing Items - Get

The

get()

method returns

None

if the key is not found.

#### Code



1

2

3

4

5

dict\_a = {

'name': 'Teja',

'age': 15

}

print(dict\_a.get('name'))

PYTHON

#### Output



Teja

#### Code



1

2

3

4

5

dict\_a = {

'name': 'Teja',

'age': 15

}

print(dict\_a.get('city'))

PYTHON

#### Output



None

### KeyError

When we use the square brackets

[]

to access the key-value, **KeyError** is raised in case a key is not found in the dictionary.

#### Code



1

2

dict\_a = {'name': 'Teja','age': 15 }

print(dict\_a['city'])

PYTHON

#### Output



KeyError: 'city'

**Quick Tip**

If we use the square brackets

[]

,

KeyError

 is raised in case a key is not found in the dictionary. On the other hand, the

get()

 method returns

None

 if the key is not found.

### Membership Check

Checks if the given key exists.

#### Code



1

2

3

4

5

6

dict\_a = {

'name': 'Teja',

'age': 15

}

result = 'name' in dict\_a

print(result)

PYTHON

#### Output



True

## Operations on Dictionaries

We can update a dictionary by

* Adding a key-value pair
* Modifying existing items
* Deleting existing items

### Adding a Key-Value Pair

#### Code



1

2

3

dict\_a = {'name': 'Teja','age': 15 }

dict\_a['city'] = 'Goa'

print(dict\_a)

PYTHON

#### Output



{'name': 'Teja', 'age': 15, 'city': 'Goa'}

### Modifying an Existing Item

As dictionaries are mutable, we can modify the values of the keys.

#### Code



1

2

3

4

5

6

dict\_a = {

'name': 'Teja',

'age': 15

}

dict\_a['age'] = 24

print(dict\_a)

PYTHON

#### Output



{'name': 'Teja', 'age': 24}

### Deleting an Existing Item

We can also use the

del

keyword to remove individual items or the entire dictionary itself.

#### Code



1

2

3

4

5

6

dict\_a = {

'name': 'Teja',

'age': 15

}

del dict\_a['age']

print(dict\_a)

PYTHON

#### Output



{'name': 'Teja'}

## Dictionary Views

They provide a dynamic view on the dictionary’s entries, which means that when the dictionary changes, the view reflects these changes.

**Dictionary Methods**

* dict.keys()
  + returns dictionary Keys
* dict.values()
  + returns dictionary Values
* dict.items()
  + returns dictionary items(key-value) pairs

The objects returned by

keys()

,

values()

&

items()

are **View Objects** .

### Getting Keys

The

keys()

method returns a view object of the type dict\_keys that holds a list of all keys.

#### Code



1

2

3

4

5

dict\_a = {

'name': 'Teja',

'age': 15

}

print(dict\_a.keys())

PYTHON

#### Output



dict\_keys(['name', 'age'])

### Getting Values

The

values()

method returns a view object that displays a list of all the values in the dictionary.

#### Code



1

2

3

4

5

dict\_a = {

'name': 'Teja',

'age': 15

}

print(dict\_a.values())

PYTHON

#### Output



dict\_values(['Teja', 15])

### Getting Items

The

items()

method returns a view object that displays a list of dictionary's (key, value) tuple pairs.

#### Code



1

2

3

4

5

dict\_a = {

'name': 'Teja',

'age': 15

}

print(dict\_a.items())

PYTHON

#### Output



dict\_items([('name', 'Teja'), ('age', 15)])

### Iterate over Dictionary Views

**Example - 1**

#### Code



1

2

3

4

5

6

dict\_a = {

'name': 'Teja',

'age': 15

}

for key in dict\_a.keys():

print(key)

PYTHON

#### Output



name

age

**Example - 2**

#### Code



1

2

3

4

5

6

dict\_a = {

'name': 'Teja',

'age': 15

}

keys\_list = list(dict\_a.keys())

print(keys\_list)

PYTHON

#### Output



['name', 'age']

**Example - 3**

#### Code



1

2

3

4

5

6

dict\_a = {

'name': 'Teja',

'age': 15

}

for value in dict\_a.values():

print(value)

PYTHON

#### Output



Teja

15

**Example - 4**

#### Code



1

2

3

4

5

6

7

dict\_a = {

'name': 'Teja',

'age': 15

}

for key, value in dict\_a.items():

pair = "{} {}".format(key,value)

print(pair)

PYTHON

#### Output



name Teja

age 15

### Dictionary View Objects

keys()

,

values()

&

items()

are called Dictionary Views as they provide a dynamic view on the dictionary’s items.

#### Code



1

2

3

4

5

6

7

8

dict\_a = {

'name': 'Teja',

'age': 15

}

view = dict\_a.keys()

print(view)

dict\_a['roll\_no'] = 10

print(view)

PYTHON

#### Output



dict\_keys(['name', 'age'])

dict\_keys(['name', 'age', 'roll\_no'])

## Converting to Dictionary

dict(sequence)

takes any number of key-value pairs and converts to dictionary.

#### Code



1

2

3

4

5

6

7

list\_a = [

("name","Teja"),

["age",15],

("roll\_no",15)

]

dict\_a = dict(list\_a)

print(dict\_a)

PYTHON

#### Output



{'name': 'Teja', 'age': 15, 'roll\_no': 15}

#### Code



1

2

3

list\_a = ["name", "Teja", 15]

dict\_a = dict(list\_a)

print(dict\_a)

PYTHON

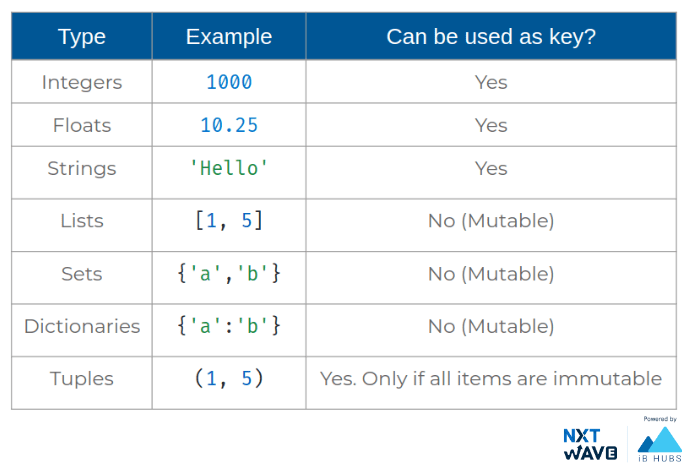
#### Output



ValueError: dictionary update sequence element #0 has length 4; 2 is required

### Type of Keys

A dictionary key must be of a type that is immutable.



Notes

Discussions

Notes

NEW NOTE

CANCELSAVE

[**Download App**](https://play.google.com/store/apps/details?id=co.ibhubs.chat)

**Personal Manager**



1

1



[Home](https://learning.ccbp.in/)Code Playground

Nakka Vijay Kumar

**NK**

**Cheat Sheet**

Cheat Sheet

**Working with Dictionaries**

Dictionary Methods

Python provides dictionary methods that allow us to work with dictionaries.

* copy()
* get()
* update()
* fromkeys()

 and more..

Let’s learn few among them

Referring Same Dictionary Object

**Code**



1

2

3

4

5

6

7

8

9

dict\_a = {

'name': 'Teja',

'age': 15

}

dict\_b = dict\_a

dict\_b['age'] = 20

print(dict\_a)

print(id(dict\_a))

print(id(dict\_b))

PYTHON

**Output**



{'name':'Teja', 'age': 20}

140170705626624

140170705626624

Copy of Dictionary

dict.copy()

returns copy of a dictionary.

**Code**



1

2

3

4

5

6

7

8

9

dict\_a = {

'name': 'Teja',

'age': 15

}

dict\_b = dict\_a.copy()

dict\_b['age'] = 20

print(dict\_a)

print(id(dict\_a))

print(id(dict\_b))

PYTHON

**Output**



{'name':'Teja', 'age': 15}

140664418952704

140664418952896

Copy of List

**Code**



1

2

3

4

5

6

list\_a = ['Teja', 15]

list\_b = list\_a.copy()

list\_b.extend([20])

print(list\_a)

print(id(list\_a))

print(id(list\_b))

PYTHON

**Output**



['Teja', 15]

139631861316032

139631860589504

Operations on Dictionaries

* len()
* clear()
* Membership Check

**Code**



1

2

3

4

5

6

7

8

9

dict\_a = {

'name': 'Teja',

'age': 15

}

print(len(dict\_a)) # length of dict\_a

if 'name' in dict\_a: # Membership Check

print("True")

dict\_a.clear() # clearing dict\_a

print(dict\_a)

PYTHON

**Output**



2

True

{}

Iterating

Cannot add/remove dictionary keys while iterating the dictionary.

**Code**



1

2

3

4

5

dict\_a = {'name': 'Teja', 'age': 15}

for k in dict\_a.keys():

if k == 'name':

del dict\_a[k]

print(dict\_a)

PYTHON

**Output**



RuntimeError: dictionary changed size during iteration

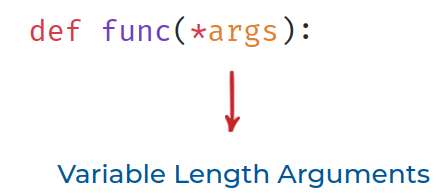
**Arbitrary Function Arguments**

Passing Multiple Values

We can define a function to receive any number of arguments.

We have already seen such functions

* max(\*args) max(1,2,3..)
* min(\*args) min(1,2,3..)



Variable Length Arguments

Variable length arguments are packed as tuple.

**Code**



1

2

3

4

5

def more\_args(\*args):

print(args)

more\_args(1, 2, 3, 4)

more\_args()

PYTHON

**Output**



(1, 2, 3, 4)

()

Unpacking as Arguments

If we already have the data required to pass to a function as a sequence, we can

unpack it with

\*

while passing.

**Code**



1

2

3

4

5

def greet(arg1="Hi", arg2="Ram"):

print(arg1 + " " + arg2)

data = ["Hello", "Teja"]

greet(\*data)

PYTHON

**Output**

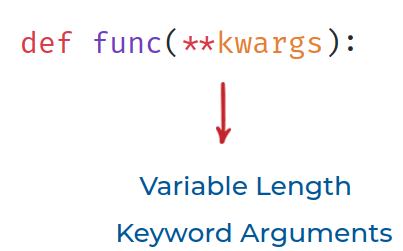


Hello Teja

Multiple Keyword Arguments

We can define a function to receive any number of keyword arguments.

Variable length kwargs are packed as dictionary.



**Code**



1

2

3

4

5

def more\_args(\*\*kwargs):

print(kwargs)

more\_args(a=1, b=2)

more\_args()

PYTHON

**Output**



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

{}

Iterating

**kwargs** is a dictionary. We can iterate over them like any other dictionary.

**Code**



1

2

3

4

5

def more\_args(\*\*kwargs):

for i, j in kwargs.items():

print('{}:{}'.format(i,j))

more\_args(a=1, b=2)

PYTHON

**Output**



a:1

b:2

Unpacking as Arguments

**Code - 1**



1

2

3

4

5

def greet(arg1="Hi", arg2="Ram"):

print(arg1 + " " + arg2)

data = {'arg1':'Hello', 'arg2':'Teja'}

greet(\*\*data)

PYTHON

**Output**



Hello Teja

**Code - 2**



1

2

3

4

5

def greet(arg1="Hi", arg2="Ram"):

print(arg1 + " " + arg2)

data = {'msg':'Hello', 'name':'Teja'}

greet(\*\*data)

PYTHON

**Output**



TypeError: greet() got an unexpected keyword argument 'msg'

Notes

Discussions

Notes

NEW NOTE

CANCELSAVE

[**Download App**](https://play.google.com/store/apps/details?id=co.ibhubs.chat)

**Personal Manager**



1