

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

Nakka Vijay Kumar

**NK**

**Cheat Sheet**

Cheat Sheet

**Tuples and Sequences**

None

None

is an object which is a datatype of its own (NoneType).

Used to define no value or nothing.

**Code**



1

2

3

var = None

print(var)

print(type(var))

PYTHON

**Output**



None

<class 'NoneType'>

Function Without Return

Functions assigned to a variable, when function does not have a

return

statement, the variable will get the value

None

**Code**



1

2

3

4

5

6

def increment(a):

a += 1

a = 55

result = increment(a)

print(result)

PYTHON

**Output**



None

Function That Returns Nothing

When a function returns no value, the default value will be

None

*Example - 1*

**Code**



1

2

3

4

5

6

7

def increment(a):

a += 1

return

a = 55

result = increment(a)

print(result)

PYTHON

**Output**



None

*Example - 2*

**Code**



1

2

3

4

5

6

7

def increment(a):

a += 1

return None

a = 5

result = increment(a)

print(result)

PYTHON

**Output**



None

*Example - 3*

**Code**



1

2

result = print("Hi")

print(result)

PYTHON

**Output**



Hi

None

**Tuple**

* Holds an ordered sequence of items.
* Tuple is immutable object, where as list is a mutable object.

**Code**



a = 2

tuple\_a = (5, "Six", a, 8.2)

Creating a Tuple

* Created by enclosing elements within (round) brackets.
* Each item is separated by a comma.

**Code**



1

2

3

4

a = 2

tuple\_a = (5, "Six", a, 8.2)

print(type(tuple\_a))

print(tuple\_a)

PYTHON

**Output**



<class 'tuple'>

(5, 'Six', 2, 8.2)

Tuple with a Single Item

**Code**



1

2

3

a = (1,)

print(type(a))

print(a)

PYTHON

**Output**



<class 'tuple'>

(1,)

Accessing Tuple Elements

Accessing Tuple elements is also similar to string and list accessing and slicing.

**Code**



1

2

3

a = 2

tuple\_a = (5, "Six", a, 8.2)

print(tuple\_a[1])

PYTHON

**Output**



Six

Tuples are Immutable

Tuples does not support modification.

**Code**



1

2

3

tuple\_a = (1, 2, 3, 5)

tuple\_a[3] = 4

print(tuple\_a)

PYTHON

**Output**



TypeError: 'tuple' object does not support item assignment

Operations can be done on Tuples

* len()
* Iterating
* Slicing
* Extended Slicing

Converting to Tuple

tuple(sequence)

Takes a sequence and converts it into tuple.

String to Tuple

**Code**



1

2

3

color = "Red"

tuple\_a = tuple(color)

print(tuple\_a)

PYTHON

**Output**



('R', 'e', 'd')

List to Tuple

**Code**



1

2

3

list\_a = [1, 2, 3]

tuple\_a = tuple(list\_a)

print(tuple\_a)

PYTHON

**Output**



(1, 2, 3)

Sequence to Tuple

**Code**



1

2

tuple\_a = tuple(range(4))

print(tuple\_a)

PYTHON

**Output**



(0, 1, 2, 3)

Membership Check

Check if given data element is part of a sequence or not.

Membership Operators

* in
* not in

*Example - 1*

**Code**



1

2

3

tuple\_a = (1, 2, 3, 4)

is\_part = 5 in tuple\_a

print(is\_part)

PYTHON

**Output**



False

*Example - 2*

**Code**



1

2

3

tuple\_a = (1, 2, 3, 4)

is\_part = 1 not in tuple\_a

print(is\_part)

PYTHON

**Output**



False

List Membership

**Code**



1

2

3

list\_a = [1, 2, 3, 4]

is\_part = 1 in list\_a

print(is\_part)

PYTHON

**Output**



True

String Membership

**Code**



1

2

3

word = 'Python'

is\_part = 'th' in word

print(is\_part)

PYTHON

**Output**



True

**Packing & Unpacking**

Unpacking

Values of any sequence can be directly assigned to variables.

Number of variables in the left should match the length of sequence.

**Code**



1

2

3

4

5

tuple\_a = ('R', 'e', 'd')

(s\_1, s\_2, s\_3) = tuple\_a

print(s\_1)

print(s\_2)

print(s\_3)

PYTHON

**Output**



R

e

d

Errors in Packing

**Code**



1

2

3

4

tuple\_a = ('R', 'e', 'd')

s\_1, s\_2 = tuple\_a

print(s\_1)

print(s\_2)

PYTHON

**Output**



ValueError: too many values to unpack (expected 2)

**Code**



1

2

3

tuple\_a = ('R', 'e', 'd')

s\_1, s\_2, s\_3, s\_4 = tuple\_a

print(s\_1)

PYTHON

**Output**



ValueError: not enough values to unpack (expected 4, got 3)

Tuple Packing

()

brackets are optional while creating tuples.

In Tuple Packing, Values separated by commas will be packed into a tuple.

**Code**



1

2

3

a = 1, 2, 3

print(type(a))

print(a)

PYTHON

**Output**



<class 'tuple'>

(1, 2, 3)

**Code**



1

2

3

a = 1,

print(type(a))

print(a)

PYTHON

**Output**



<class 'tuple'>

(1,)

**Code**



1

2

a, = 1,

print(type(a)) print(a)

PYTHON

**Output**



<class 'int'>

1

Notes

Discussions

Notes

NEW NOTE

CANCELSAVE

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

**Personal Manager**



1



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

Nakka Vijay Kumar

**NK**

**Cheat Sheet**

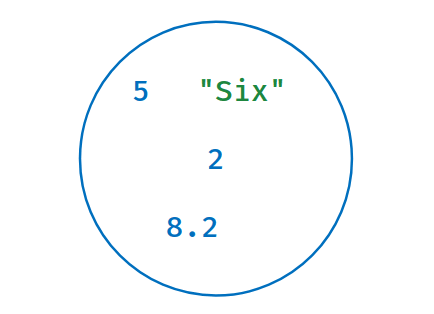
Cheat Sheet

**Sets**

Unordered collection of items.

Every set element is

* Unique (no duplicates)
* Must be immutable



Creating a Set

* Created by enclosing elements within *{curly}* brackets.
* Each item is separated by a comma.

**Code**



1

2

3

4

a = 2

set\_a = {5, "Six", a, 8.2}

print(type(set\_a))

print(set\_a)

PYTHON

**Output**



<class 'set'>

{8.2, 2, 'Six', 5}

Need not be in the same order as defined.

No Duplicate Items

Sets contain unique elements

**Code**



1

2

set\_a = {"a", "b", "c", "a"}

print(set\_a)

PYTHON

**Output**



{'b', 'a', 'c'}

Immutable Items

Set items must be immutable. As List is mutable, Set cannot have list as an item.

**Code**



1

2

set\_a = {"a", ["c", "a"]}

print(set\_a)

PYTHON

**Output**



TypeError: unhashable type: 'list'

Creating Empty Set

We use

set()

to create an empty set.

**Code**



1

2

3

set\_a = set()

print(type(set\_a))

print(set\_a)

PYTHON

**Output**



<class 'set'>

set()

Converting to Set

set(sequence)

takes any sequence as argument and converts to set, avoiding duplicates

List to Set

**Code**



1

2

3

set\_a = set([1,2,1])

print(type(set\_a))

print(set\_a)

PYTHON

**Output**



<class 'set'>

{1, 2}

String to Set

**Code**



1

2

set\_a = set("apple")

print(set\_a)

PYTHON

**Output**



{'l', 'p', 'e', 'a'}

Tuple to Set

**Code**



1

2

set\_a = set((1, 2, 1))

print(set\_a)

PYTHON

**Output**



{1, 2}

Accessing Items

As sets are unordered, we cannot access or change an item of a set using

* Indexing
* Slicing

**Code**



1

2

3

set\_a = {1, 2, 3}

print(set\_a[1])

print(set\_a[1:3])

PYTHON

**Output**



TypeError: 'set' object is not subscriptable

Adding Items

set.add(value)

adds the item to the set, if the item is not present already.

**Code**



1

2

3

set\_a = {1, 3, 6, 2, 9}

set\_a.add(7)

print(set\_a)

PYTHON

**Output**



{1, 2, 3, 6, 7, 9}

Adding Multiple Items

set.update(sequence)

adds multiple items to the set, and duplicates are avoided.

**Code**



1

2

3

set\_a = {1, 3, 9}

set\_a.update([2, 3])

print(set\_a)

PYTHON

**Output**

{2, 1, 3, 9}

Removing Specific Item

set.discard(value)

takes a single value and removes if present.

**Code**



1

2

3

set\_a = {1, 3, 9}

set\_a.discard(3)

print(set\_a)

PYTHON

**Output**



{1, 9}

set\_a.remove(value)

takes a value and remove if it present or raise an error.

**Code**



1

2

3

set\_a = {1, 3, 9}

set\_a.remove(5)

print(set\_a)

PYTHON

**Output**

KeyError: 5

Operations on Sets

You can perform the following operations on Sets

* clear()
* len()
* Iterating
* Membership Check

Notes

Discussions

Notes

NEW NOTE

CANCELSAVE

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

**Personal Manager**



1



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

Nakka Vijay Kumar

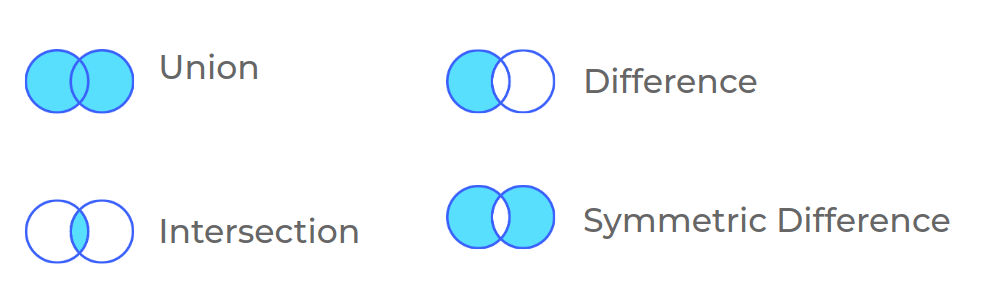
**NK**

**Cheat Sheet**

Cheat Sheet

**Set Operations**

Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.



Union

Union of two sets is a set containing all elements of both sets.

set\_a | set\_b

or

set\_a.union(sequence)

union()

converts sequence to a set, and performs the union.

**Code**



1

2

3

4

set\_a = {4, 2, 8}

set\_b = {1, 2}

union = set\_a | set\_b

print(union)

PYTHON

**Output**



{1, 2, 4, 8}

**Code**



1

2

3

4

set\_a = {4, 2, 8}

list\_a = [1, 2]

union = set\_a.union(list\_a)

print(union)

PYTHON

**Output**



{1, 2, 4, 8}

Intersection

Intersection of two sets is a set containing common elements of both sets.

set\_a & set\_b

or

set\_a.intersection(sequence)

intersection()

converts sequence to a set, and perform the intersection.

**Code**



1

2

3

4

set\_a = {4, 2, 8}

set\_b = {1, 2}

intersection = set\_a & set\_b

print(intersection)

PYTHON

**Output**



{2}

**Code**



1

2

3

4

set\_a = {4, 2, 8}

list\_a = [1, 2]

intersection = set\_a.intersection(list\_a)

print(intersection)

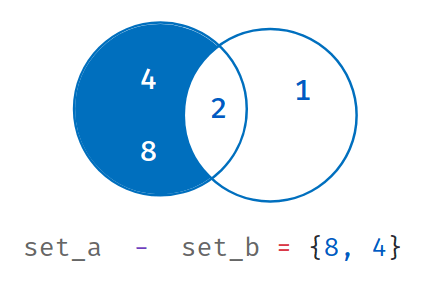
PYTHON

**Output**



{2}

Difference of two sets is a set containing all the elements in first set but not second.



set\_a - set\_b

or

set\_a.difference(sequence)

difference()

converts sequence to a set.

**Code**



1

2

3

4

set\_a = {4, 2, 8}

set\_b = {1, 2}

diff = set\_a - set\_b

print(diff)

PYTHON

**Output**



{8, 4}

**Code**



set\_a = {4, 2, 8}

tuple\_a = (1, 2)

diff = set\_a.difference(tuple\_a)

print(diff)

**Output**



{8, 4}

Symmetric Difference

Symmetric difference of two sets is a set containing all elements which are not common to both sets.

set\_a ^ set\_b

or

set\_a.symmetric\_difference(sequence)

symmetric\_difference()

converts sequence to a set.

**Code**



1

2

3

4

set\_a = {4, 2, 8}

set\_b = {1, 2}

symmetric\_diff = set\_a ^ set\_b

print(symmetric\_diff)

PYTHON

**Output**



{8, 1, 4}

**Code**



1

2

3

4

set\_a = {4, 2, 8}

set\_b = {1, 2}

diff = set\_a.symmetric\_difference(set\_b)

print(diff)

PYTHON

**Output**



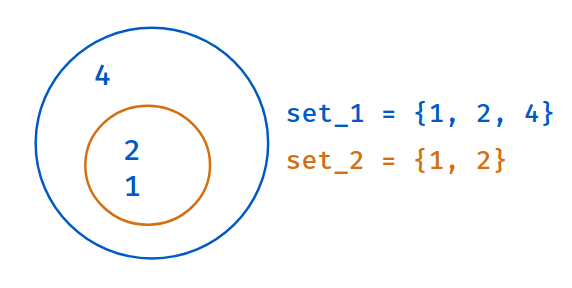
{8, 1, 4}

**Set Comparisons**

Set comparisons are used to validate whether one set fully exists within another

* issubset()
* issuperset()
* isdisjoint()

Subset



set2.issubset(set1)

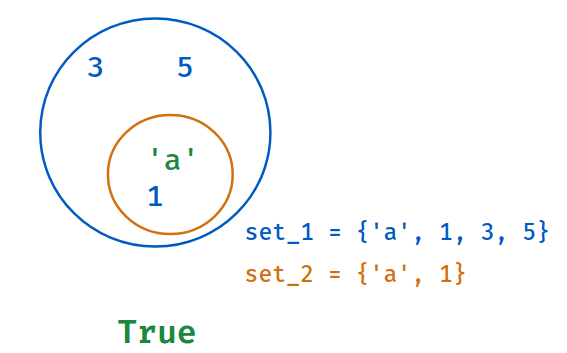
Returns

True

if all elements of second set are in first set. Else,

False

*Example - 1*



**Code**



1

2

3

4

set\_1 = {'a', 1, 3, 5}

set\_2 = {'a', 1}

is\_subset = set\_2.issubset(set\_1)

print(is\_subset)

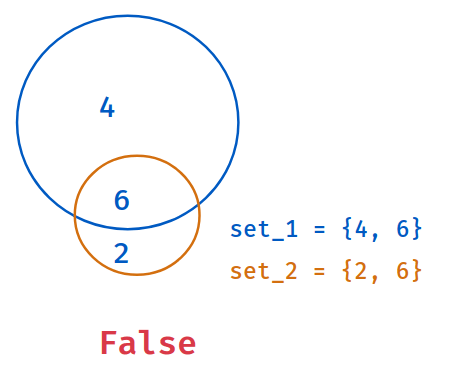
PYTHON

**Output**



True

*Example - 2*



**Code**



1

2

3

4

set\_1 = {4, 6}

set\_2 = {2, 6}

is\_subset = set\_2.issubset(set\_1)

print(is\_subset)

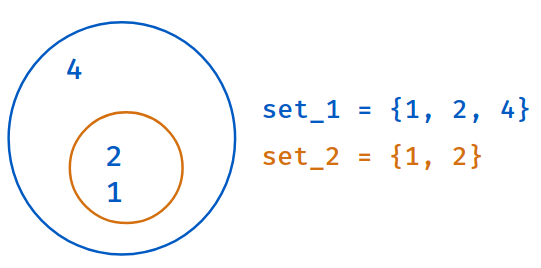
PYTHON

**Output**



False

SuperSet



set1.issuperset(set2)

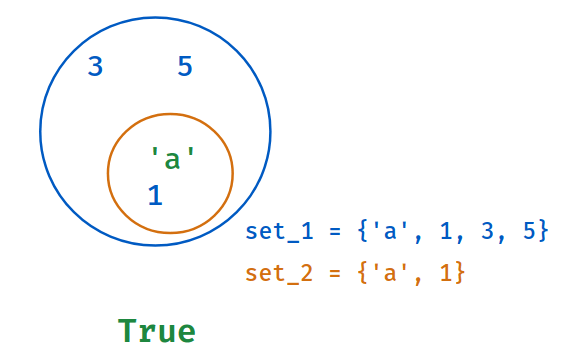
Returns

True

if all elements of second set are in first set. Else,

False

*Example - 1*



**Code**



1

2

3

4

set\_1 = {'a', 1, 3, 5}

set\_2 = {'a', 1}

is\_superset = set\_1.issuperset(set\_2)

print(is\_superset)

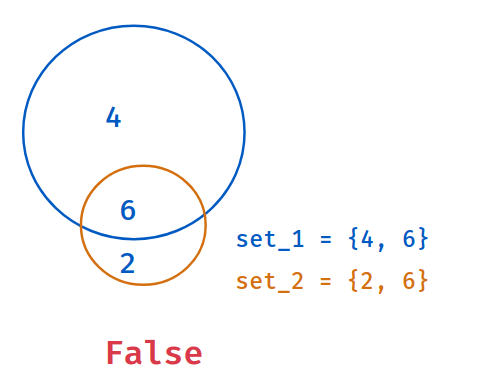
PYTHON

**Output**



True

*Example - 2*



**Code**



set\_1 = {4, 6}

set\_2 = {2, 6}

is\_superset = set\_1.issuperset(set\_2)

print(is\_superset)

**Output**



False

Disjoint Sets

set1.isdisjoint(set2)

Returns

True

when they have no common elements. Else,

False

**Code**



1

2

3

4

set\_a = {1, 2}

set\_b = {3, 4}

is\_disjoint = set\_a.isdisjoint(set\_b)

print(is\_disjoint)

PYTHON

**Output**



True

Notes

Discussions

Notes

NEW NOTE

CANCELSAVE

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

**Personal Manager**



1