# SETS

A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets.

Every set element is unique (no duplicates) and must be immutable (cannot be changed).

However, a set itself is mutable. We can add or remove items from it.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

- A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
 # set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

**Output**

```
{1, 2, 3}
{1.0, (1, 2, 3), 'Hello'}
```

```
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)
my_set = set([1, 2, 3, 2])
 print(my_set)
```

**Output**

```
{1, 2, 3,4}
{1, 2, 3}
```

- set cannot have mutable items # here [3, 4] is a mutable list # this will cause an error.
- my_set = {1, 2, [3, 4]}

- Output: ERROR

## Creating an empty set

- Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements, we use the set() function without any argument.
- # Distinguish set and dictionary while creating empty set
- # initialize a with {}
- a = {}
- # check data type of a print(type(a))
- # initialize a with set()
- a = set()
- # check data type of a print(type(a))

**Output**

```
<class 'dict'>
<class 'set'>
```

## Modifying a set in Python

- Sets are mutable. However, since they are unordered, indexing has no meaning.
- We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.
- We can add a single element using the add() method, and multiple elements using the update() method.
- The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
# initialize my_set
my_set = {1, 3}
 print(my_set)
TypeError: 'set' object does not support
   indexing
# my_set[0]
```

```
#add an element
my_set.add(2)
 print(my_set)
 # add multiple elements
 my_set.update([2, 3, 4])
print(my_set)
# add list and set
my_set.update([4, 5], {1, 6, 8})
print(my_set)
```

**Output**

```
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

## Removing elements from a set

- A particular item can be removed from a set using the methods **discard() and remove().**
- The only difference between the two is that the discard() function leaves a set unchanged if the element is not present in the set.
- On the other hand, the remove() function will raise an error in such a condition (if element is not present in the set).

## Discard()

- **# initialize my_set**
- my_set = {1, 3, 4, 5, 6}
- print(my_set)
- **# discard an element**
- # Output: {1, 3, 5, 6}
- my_set.discard(4)
- print(my_set)

```
# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
 print(my_set)
```

```
# discard an element not present
  in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)
```

# remove an element not present in my_set  you will get an error.

**# Output: KeyError**

my_set.remove(2)

- Similarly, we can remove and return an item using the **pop()** method.
- Since set is an unordered data type, there is no way of determining which item will be popped. It is completely arbitrary.
- We can also remove all the items from a set using the clear() method.

```
# initialize my_set
my_set = set("HelloWorld")
print(my_set)
 # pop an element
print(my_set.pop())
 # pop another element
my_set.pop()
 print(my_set)
 # clear my_set
  my_set.clear()
 print(my_set)
 print(my_set)
```

**Output**

```
{'H', 'l', 'r', 'W', 'o', 'd','e'}
H
 {'r', 'W', 'o', 'd', 'e'}
 set()
```

A set is created by using the set() function or placing all the elements within a pair of curly braces.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
Months={"Jan","Feb","Mar"}
 Dates={21,22,17}
print(Days)
 print(Months)
 print(Dates)
```

## Accessing Values in a Set

- We cannot access individual values in a set. We can only access all the elements together as shown above. But we can also get a list of individual elements by looping through the set.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
for d in Days:
   print(d)
```

```
Sun
Wed
Fri
Tue
Mon
Thu
Sat
```

## Get the Length of a Set

To determine how many items a set has, use the len() method.

**Example**

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

OUTPUT:

```
3
```

# Set Methods

- add()Adds an element to the set
- clear()Removes all the elements from the set
- copy()Returns a copy of the set
- difference()Returns a set containing the difference between two or more sets
- Difference_update()Removes the items in this set that are also included in another, specified set
- discard()Remove the specified item

- intersection()Returns a set, that is the intersection of two other sets
- intersection_update()Removes the items in this set that are not present in other, specified set(s)
- isdisjoint()Returns whether two sets have a intersection or not
- issubset()Returns whether another set contains this set or not
- issuperset()Returns whether this set contains another set or not
- pop()Removes an element from the set
- remove()Removes the specified element
- symmetric_difference()Returns a set with the symmetric differences of two sets
- symmetric_difference_update()inserts the symmetric differences from this set and another
- union()Return a set containing the union of sets
- update()Update the set with the union of this set and others