Python Sets

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).
However, the set itself is mutable. We can add or remove items from it.
Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

How to create a set?

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

Python Lists Vs array Module as Arrays

We can treat lists as arrays. However, we cannot constrain the type of elements stored in a list. For example:

1.
2. a = [1, 3.5, "Hello"]

If you create arrays using the array module, all elements of the array must be of the same numeric type.

```
import array as arr

a = arr.array('d', [1, 3.5, "Hello"])   // Error

my_set = {1, 2, 3}
print(my_set)

my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Another example if you will

```
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
```

```
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# If you uncomment line #12,
# this will cause an error.
# TypeError: unhashable type: 'list'

#my_set = {1, 2, [3, 4]}

# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1,2,3,2])
print(my_set)
```

reating an empty set is a bit tricky.
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.

```
a = {}

# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()

# check data type of a
# Output: <class 'set'>
print(type(a))
```

How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.
We cannot access or change an element of set using indexing or slicing. Set does not support it.
We can add 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.

```
my_set = {1,3}
```

```
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

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

How to remove elements from a set?

A particular item can be removed from set using methods, discard() and remove().
The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.
The following example will illustrate this.

```
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
# If you uncomment line 27,
# you will get an error.
# Output: KeyError: 2

#my_set.remove(2)
```

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

```
# initialize my_set
# Output: set of unique elements
my_set = set("HelloWorld")
print(my_set)

# pop an element
# Output: random element
print(my_set.pop())

# pop another element
# Output: random element
my_set.pop()
print(my_set)

# clear my_set
#Output: set()
my_set.clear()
print(my_set)
```

Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.
Let us consider the following two sets for the following operations.

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

You can also use the keywords

```
A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}

# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Set Intersection

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

Set diferences

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```