A Python set object is a collection of distinct objects. A set is mutable — the contents can be changed using methods like `add()` and `remove()`. Like other collections, a set is iterable. But there is no notion of element position or order. Accordingly, sets do not support indexing, slicing, or other sequence-like behavior.

- `set(iterable_exp)`: creates and returns a set containing all the objects in the argument `iterable_exp`, which itself must be iterable.
- `{exp₁, exp₂, ..., expₙ}`: creates and returns a set containing the values returned by `exp₁, exp₂, ..., expₙ`.
- `len(s)`: returns the cardinality of (number of objects in) set `s`.
- `x in s`: tests whether set `s` contains `x` as one of its objects.
- `x not in s`: tests whether set `s` does not contain `x` as one of its objects.
- `s.isdisjoint(other)`: tests whether the sets `s` and `other` have no elements in common.
- `s.issubset(other), s <= other`: tests whether every element in set `s` is also in set `other`.
- `s < other`: tests whether the set `s` is a proper subset of the set `other`.
- `s.issuperset(other), s >= other`: tests whether every element in the set `other` is in the set `s`.
- `s > other`: tests whether the set `s` is a proper superset of the set `other`.
- `s == other`: tests whether the sets `s` and `other` have the same contents.
- `s.union(other, ...), s | other | ...`: return the union (a new set) of the set `s` and all the `other` sets.
- `s.intersection(other, ...), s & other & ...`: Return the intersection (a new set) of the set `s` and all the `other` sets.
- `s.difference(other, ...), s - other - ...`: returns a new set with elements in the set `s` that are not in any of the `other` sets.
- `s.symmetric_difference(other), set ^ other`: returns a new set with elements in either of the sets `s` or `other` but not in both.
• `s.copy()`: returns a new set which is a shallow copy of `s`.

• `s.add(exp)`: adds `exp` to the set `s` *(in place)*. (Returns None.)

• `s.remove(exp)`: removes `exp` from the set `s` *(in place)*. Raises `KeyError` if elem is not contained in `s`.

• `s.discard(exp)`: removes `exp` from set `s` if it is present; otherwise, does nothing.

• `s.pop()`: removes and returns an arbitrary element from `s`. Raises `KeyError` if `s` is empty.

• `s.clear()`: removes all elements from set `s` *(in place)*.

• `for v in s:`
  ```
  suite
  ```

  iterates through the elements of `s`, assigning each element to `v` and executing `suite` after each assignment.