Sets in Python

In Python, a **set** is a collection of items, similar to a **list**. Like lists, sets can contain items of the same type, such as all integers, or a mix of different types, like integers and strings.

However, there are key differences between sets and lists:

- **Distinct Items:** A set only contains unique items. This means no two items in a set can be the same. For example, if you have a set *s = {10, 20, 30}* and try to add *20* again, the set remains unchanged.
- **Unordered:** Sets do not maintain any particular order of items. If you add items in a specific order and then print the set, the items may appear in any order.
- **No Indexing:** Unlike lists, sets do not support indexing. You cannot access items by their position in the set.

Why Use Sets?

Despite these differences, sets offer several advantages:

- **Fast Operations:** Operations like *union*, *intersection*, and *difference* are performed quickly on sets.
- **Hashing:** Internally, sets use **hashing**, which allows for fast search, insert, and delete operations.

Creating Sets in Python

You can create a set using curly braces `{}` or the **set()** constructor.

```
s1 = {10, 20, 30}
print(s1)        # Output: {10, 20, 30}
```

```
s2 = set([20, 30, 40])
print(s2)        # Output: {40, 20, 30}
```

```
s3 = {}
print(type(s3))      # Output: <class 'dict'>
```

```
s4 = set()
print(type(s4))   # Output: <class 'set'>
print(s4)         # Output: set()
```

Note that using empty curly braces `{}` creates an empty **dictionary**, not a set. To create an empty set, use **set()**.

Adding Elements to a Set

```
s = {10, 20}
s.add(30)
print(s)      # Output: {10, 20, 30}
```

```
s.add(30)
print(s)      # Output: {10, 20, 30}
```

```
s.update([40, 50])
print(s)      # Output: {40, 10, 50, 20, 30}
```

The **add()** method adds a single item to the set. If the item already exists, the set remains unchanged. The **update()** method can add multiple items from another collection, such as a list or another set.

Removing Elements from a Set

```
s = {10, 30, 20, 40}
s.discard(30)
```

```
print(s)       # Output: {10, 20, 40}
```

```
s.remove(20)
print(s)       # Output: {40, 10}
```

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

```
s.add(50)
del s
```

```
After del s, accessing s will raise an error
```

- **discard()**: Removes an item if it exists. If the item is not present, it does nothing.
- **remove()**: Removes an item if it exists. If the item is not present, it raises an error.
- **clear()**: Removes all items from the set, resulting in an empty set.
- **del** statement: Deletes the entire set object. Attempting to access it afterward will result in an error.

Other Set Operations

```
s = {10, 30, 20, 40}
print(len(s))       # Output: 4
print(20 in s)      # Output: True
print(50 in s)      # Output: False
```

- **len()**: Returns the number of items in the set.
- **in** operator: Checks if an item is present in the set.

These operations are faster on sets compared to lists due to the underlying hashing mechanism.

Set Operations: Union, Intersection, Difference

```
s1 = {2, 4, 6, 8}
s2 = {3, 6, 9}
```

```
print(s1 | s2)        # Output: {2, 3, 4, 6, 8, 9}
print(s1 & s2)        # Output: {6}
print(s1 - s2)        # Output: {2, 4, 8}
print(s1 ^ s2)        # Output: {2, 3, 4, 8, 9}
```

- **Union (`|`):** Combines all unique elements from both sets.
- **Intersection (`&`):** Returns only the common elements between the sets.
- **Difference (`-`):** Returns elements present in the first set but not in the second.
- **Symmetric Difference (`^`):** Returns elements present in either set but not in both.

These operations can also be performed using methods like **union()**, **intersection()**, and **difference()**.

Subset and Superset Operations

```
s1 = {2, 4, 6, 8}
s2 = {4, 8}
```

```
print(s1.isdisjoint(s2))  # Output: False
print(s1 <= s2)           # Output: False
print(s1 < s2)            # Output: False
print(s1 >= s2)           # Output: True
print(s1 > s2)            # Output: True
```

- **isdisjoint():** Returns True if the two sets have no elements in common.
- **Subset (<=):** Checks if all elements of the first set are in the second set.
- **Proper Subset (<):** Similar to subset but does not allow both sets to be equal.
- **Superset (>=):** Checks if the first set contains all elements of the second set.
- **Proper Superset (>):** Similar to superset but does not allow both sets to be equal.