Sets Internal Working

Python

What is Set ?

A set is an unordered collection of unique elements.

It means:

- You cannot have duplicate elements.
- The order is not guaranteed (because it uses a hash table).
- It provides fast lookups, insertions, and deletions.

s = {10, 20, 30,40,50,60,70,80}

How sets store in python ?

A Python set is implemented using a hash table. But what is a hash table ?

Think of a hash table like a bookshelf :

Imagine you have a bookshelf with 8 slots, and you want to store the numbers 10, 20, 30. But instead of putting them in order, Python finds a random-looking position based on a formula (hash function).

How does Python decide where to put each number ?

Computes a hash :

Python uses the hash() function to generate a number.

hash(56)

It's contain only immutable data like (string, number, boolean, tuple)

Finds a bucket (position) :

Uses modulus (%) with table size to get an index.

Formula :

hash(value) % slots = position

The numbers will store according to the position.
For example :

hash(10) % 8 = 2

hash(20) % 8 = 4

hash(30) % 8 = 6

That's how , the values is store in the sets. Here , I want to show you a table , that you will understand how it has stored :

Table :

Position	Stored Elements
0	None
1	None
2	10
3	None
4	20
5	None
6	30
7	None

I Know, you have some queries about the slots and the values and that is , what if we have more values than 8 or less . Because 8 slots will not carry more than 8 values , so slots can change according to how many values sets have . You don’ t need to take stress, how slots will change. On the next page , we will see how the slots will change.

Why does Python use 8 slots? Why not 16,13?

Python does not always use exactly 8 slots. The number of slots (buckets) in a hash table depends on the implementation and can change dynamically when more elements are added or removed.

- Python starts with a default size (usually a power of 2 like 4, 8, 16, 32, etc.).
- If the set grows larger, Python doubles the size when necessary to maintain efficiency.

So, the number of slots could be 8, 16, 32, etc., depending on the number of elements in the set. As more elements are added, Python resizes the table, making lookups still fast.

How Does Python Increase the Number of Slots in a set?

Python dynamically increases the number of slots based on its internal logic, not just the number of values.

Slots Are Managed by Python's Internal Hash Table :

- The number of slots does not increase immediately after every new element.

- Instead, Python uses load factors to decide when to resize the hash table.

When the table reaches a certain threshold, Python doubles the number of slots to reduce collisions.

What Triggers Slot Increase ?

Python increases the slots when the number of elements grows and the table gets too full.

Load Factor : The Key Rule

Python sets have a load factor, which controls when the hash table resizes:

- Load Factor = number of values/ slots (0.66 or ~66%)
- If the number of elements exceeds 66% of the available slots, Python doubles the table size.

Example: How Slots Grow

Let's say Python starts with 8 slots:

```
s = set()
```

Step 1: Add Some Elements

```
s.add(10)
s.add(20)
s.add(30)
s.add(40)
s.add(50)
```

Python originally allocated 8 slots. Now, 5 elements are inside.

Load factor = 5 / 8 = 62.5% → Still below 66%, so no resize yet.

Step 2: Add More Elements

```
s.add(60)  # 6 elements  now
```

Load factor = 6 / 8 = 75% → Exceeds 66%!
Python now resizes the table (doubles slots to 16).

What Happens When Slots Double ?

Python creates a new table with double the size (e.g., from 8 → 16 slots).

It rehashes all existing elements into the new table.

New elements go into their new positions based on hash(value) % newSize.

This ensures faster lookups and fewer collisions.

Does the Slot Size Depend on Values ?

- No, slots don't increase just because of the values themselves.
- Slots increase based on the number of elements and the load factor.

Summary :

Number of Elements	Initial Slots	Load Factor	Resize Trigger?	New Slots
0	0	0%	No	0
5	8	62.5%	No	8
6	8	75%	Yes	16
12	16	75%	Yes	32
24	32	75%	Yes	64

Is 8 the Default Slot Size for a Python set?

- No, the default number of slots in a Python sets is NOT always 8.
- It starts with 0 slots and grows dynamically as elements are added.

How Does Python Allocate Slots Initially ?

- When you create an empty set(), it has 0 slots (not 8).
- As soon as you add the first element, Python allocates slots based on internal rules.

What Is the Initial Slot Count ?

- When you add the first element, Python does NOT start with 8 slots.
- The exact initial size depends on the Python version and implementation, but it's often around 4 or 8.

Slot Growth Example :

Elements	Approx. Slots Allocated
0	0 slots
1	4 slots
6	8 slots
12	16 slots
24	32 slots

So, when you add an element or delete an element, the slots will automatically resize, when it’s reach a certain load factor. This process is known as Rehashing.

During rehashing, all elements get redistributed, meaning their storage positions can change, even if no explicit sorting is done.

Why Does the Order Change ?

- New elements may trigger rehashing, leading to reallocation of storage.
- Removing an element can also affect the layout, especially if rehashing is triggered due to shrinking.

Does This Mean Sets Are Ordered Internally ?

No ,

Even though elements are stored based on their hash values, this does not mean sets are ordered in the way lists or tuples are. Order can change unexpectedly, so it's unreliable for ordered operations like indexing.

Work Flow :

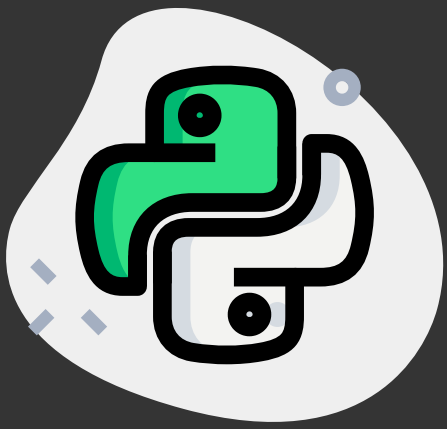
Values:	Hash Code:	% Slots :	Hash Table :
Hello	6114456990735492172		
World	5929736463341838882	% 8	
Hi	1880285580866074395		
Bye	1604238216702342326		

Index	Item
0	None
1	None
2	Hello
3	Hi
4	Bye
5	None
6	World
7	None

THAT'S THE INTERNAL WORKING OF SETS

IN PYTHON

i hope, you have learn something from here. If you want detail about the topics, so tell I will respond you soon



MADE BY SHAIKH ABDUL MOIZ

THANK YOU