**1.**

**Hash collisions handling Python set ek hash table use karta hai. Pehle element ka hash calculate hota hai. Agar do elements ka hash same ho (collision), to Python \_\_eq\_\_() check karta hai. Agar same hain to ek hi rakhta hai, warna dono alag alag jagah store karta hai. is saval ka bara ma bata o**

1. **Set ke andar har element ka hash banta hai**
   * Jaise tumne ek number ya string dala → Python uska ek **hash number** banata hai.
   * Ye hash number decide karta hai ke element memory ke kis box (bucket) me jayega.
2. **Collision ka matlab**
   * Kabhi kabhi do alag elements ka hash number **same** nikal aata hai.
   * Isko hi **collision** bolte hain.
3. **Python ka solution**
   * Jab collision hota hai to Python confuse nahi hota.
   * Pehle hash check karta hai, agar dono same ho jaye → fir **\_\_eq\_\_()** method call karta hai (equal hain ya nahi).
4. **Result**
   * Agar equal nikle → dono same samjhe jate hain → set me ek hi rakha jata hai.
   * Agar equal na hue → dono alag entry ban jate hain → set dono ko rakhta hai

### **2.**

### **Set elements hashable kyun hone chahiye?**

Set ko internally hashing ki zaroorat hoti hai taake element ka fast lookup ho.

* Agar element hashable na ho (mutable cheez jaise list), to uski hash value change ho sakti hai aur set corrupt ho jaata.
* Is liye sirf immutable (hashable) cheezein allow hain jaise tuple, string, numbers.

1. **Hash Table ka Concept**
   * Python ka set internally **hash table** use karta hai.
   * Jab bhi tum ek element set me daalte ho, uska **hash value** calculate hota hai (\_\_hash\_\_() method se).
   * Ye hash value decide karta hai ke element memory ke kaunse "bucket" me store hoga.
2. **Collision Kya Hai?**
   * Collision tab hota hai jab **do alag elements ka hash value same nikal aata hai**.
   * Example: hash("abc") aur hash("xyz") agar ek jaisa aajaye, to dono ek hi bucket me jayenge.
3. **Collision Handling in Set**
   * Agar do elements ka **hash same ho** → Python unko **\_\_eq\_\_() se compare** karta hai.
   * Agar \_\_eq\_\_() True return kare → iska matlab dono same object hain → set me duplicate allow nahi hota → sirf ek rakha jayega.
   * Agar \_\_eq\_\_() False return kare → matlab objects alag hain → dono set ke andar store kiye jayenge, bas alag "slot" me.

### **3.**

### **Garbage collection aur memory**

Badi sets jab delete karte ho to Python ke garbage collector unko memory se hata deta hai.

* Agar tum explicitly gc.collect() use karo to turant memory reclaim ho jaati hai

### **3. Garbage Collection aur Memory**

Jab hum Python me ek **badi set** banate hain, to uske elements memory occupy karte hain.  
Agar hum us set ko delete kar dein (del statement se), to Python ka **garbage collector** us memory ko free kar deta hai.

Kabhi kabhi memory turant free nahi hoti. Aise case me agar hum **gc.collect()** function explicitly call karen, to garbage collector forcefully chal jaata hai aur memory **immediately reclaim** ho jaati hai.

👉 Iska matlab hai ke large sets ke sath kaam karte waqt garbage collection memory management me important role play karti hai.

### **4.**

### **WeakSet ka advantage**

Normal set object ko strong reference se hold karta hai.

* Agar object delete karna ho to set me phir bhi pada rahega.
* WeakSet me agar asli object delete ho jaye to WeakSet automatically clean ho jaata hai → memory leak nahi hoti.

### **4. WeakSet ka Advantage**

Python ka **normal set** objects ko strong reference ke saath hold karta hai. Matlab agar hum ek object ko delete kar dein, to bhi uska reference set me maujood rahta hai jab tak manually remove na karein. Isse kabhi kabhi **memory leak** ho sakti hai.

Lekin **WeakSet** weak references rakhta hai. Agar asli object delete ho jaye, to WeakSet us object ko automatically hata deta hai.  
👉 Is tarah **memory efficient** hota hai aur objects ke unnecessary references store nahi rehte.

### **5.**

### **Set lookup vs List lookup**

* Set: O(1) (direct hash table se access).
* List: O(n) (har element check hota hai).  
  Is liye set ka lookup bohot fast hota hai.

### **5. Set Lookup vs List Lookup**

Python me **set** internally hash table use karta hai. Jab hum kisi element ka lookup karte hain (jaise x in set), to wo directly hash value check karta hai. Iska time complexity **O(1)** hota hai — matlab constant time me result mil jaata hai.

Dusri taraf, **list** sequential search karti hai. Matlab list ke har element ko ek ek karke check karna padta hai jab tak match na mil jaye. Iska time complexity **O(n)** hota hai — jitni badi list hogi utna zyada time lagega.