Python Data Structures

* Lists
* Tuples
* Sets

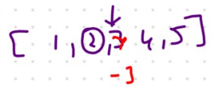
List: Ordered, mutable collection of items to be stored

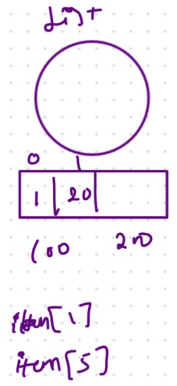
Creation:

```
            0  1  2  3  4
my_list = [1, 2, 3, 4, 5]
```

Key Operations

① Accessing elements [Zero-indexed]





** Note (Important Points)

→ Lists are mutable (can be modified after creation)

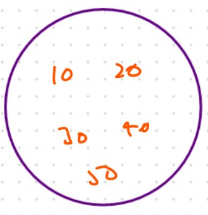
→ Lists maintain insertion order

→ Lists can contain items of different types

→ Lists can contain duplicate elements

Tuples

* Ordered, immutable collections of items

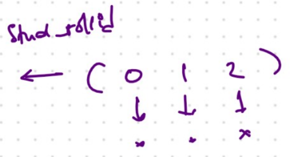


Creation

```
my_tuple = (1, 2, 3, 4)
```

Important Points

* Tuples are immutable (cannot be modified after creation)
* Tuples maintain insertion order
* Tuples can contain items of different types
* Tuples can contain duplicate elements
* Tuples are typically faster than lists
* Tuples can be used as dictionary keys (list cannot)



Sets

* Unordered, mutable collection of unique items

Creation [non indexed rather hashed]   [, (, {

my_set = {1, 2, 3, 4, 5}

Hash value

"Alex" → #0124

{"alex", "bob", "45", "charlie"}

#0127 #2912 #3012 #4012

"Alex"
#0124

{2, 1, 4, 3, 5}

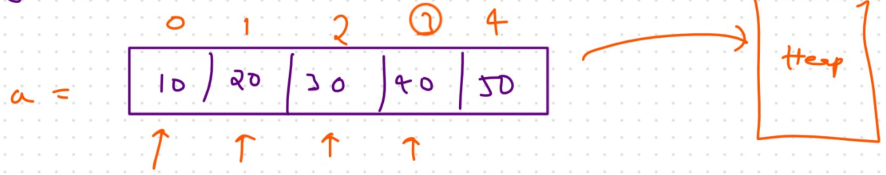
{1, 2, 3, 4, 5}

{2, 1, 3, 4, 5}

Important Points

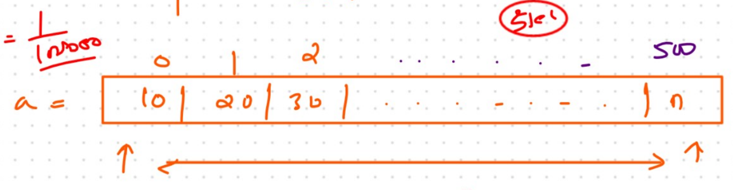
* Sets are unordered (elements have no index)
* sets only contain unique elements (duplicates are automatically removed)
* Sets can only contain hashable objects
* Sets are mutable
* sets are highly optimized for membership testing, unions, intersections

Optimization



point (a[3])

3.4 2.2 GHz CPU 1sec = 1/10000



Unit of time ⟶ 1ms, 1sec, 5sec } CPU, RAM

hash key

Hash key	value
#012	10
#013	20
#014	30
#01	40

{10, 20, 30, 40, ... - - - }

10 in my_set // True

5k

↑

#012 ⟶ 10