* **Container:**
  + Container is an entity that contains multiple data items.
  + Also known as a collection or a compound data type
* **Container Data Types:**
  + **Lists**
  + **Tuples**
  + **Sets**
  + **Dictionaries**
* **Sets**
  + Sets are like lists, with an exception that they do not contain duplicate entries.
    - a = set() 🡨 empty set, use ( ) instead of { }.
    - b = { 20 } 🡨 set with one item.
    - c = { ‘ Darshit’, 51, ‘18/02/1972’ } 🡨 set with multiple items.
    - d = { 10, 10, 10, 10,10 } 🡨 only one 10 gets stored.
  + Unordered collection
  + While storing an element in a set, its hash value is computed using a hashing technique to determine where it should be stored in the set.
  + Since hash value of an element will always be same, no matter in which order we insert the elements in a set, they get stored in the same order.
    - s = { 12, 23, 45, 16, 52 }
    - t = { 16, 52, 12, 23, 45 }
    - u = { 52, 12, 16, 45, 23 }
    - print (s)
      * {16, 52, 23, 12, 45}
    - print (t)
      * {16, 52, 23, 12, 45}
    - print (u)
      * {16, 52, 23, 12, 45}
  + It is possible to create a set of numbers, strings and tuples, but not a set of lists.
    - s1 = { 'Morning', 'Evening' } 🡨 works
    - s2 = { (12, 23 ), (15, 25 ), ( 17, 34 ) } 🡨 works
    - s3 = { [12, 23 ], [15, 25 ], [ 17, 34 ] } 🡨 Error: unhashable type: 'list'
  + Since strings and tuples are immutable, their hash value remains same at all times. Hence, a set of strings or tuples is permitted. However, a list may change, so its hash value may change, hence a set of lists is not permitted.
* **Accessing Set Elements**
  + - print (s1) 🡨 entire set can be printed by just using the name of the set.
    - print (s1[0]) 🡨 Individual elements can’t be accessed using

indices like we used earlier in strings/list.

* + - print s1[1:4] 🡨 Unlike strings & lists, sets can’t be sliced.
* **Looping in Sets**
  + Using **for** only
  + #using for loop #using enumerate() to keep track of access order and not

Insertion order

for v in s1: for index, v in enumerate(s1):

print (v) print (index, v)

* **Basic Set Operations**
  + **Set object is mutable like a list (changeable), elements can be added in a set.**
    - names = { 'Darshit', 'Ragi', 'Aashna' }
    - names.add( 'Sharda' )
  + **Set element is immutable like a tuple (element can’t be updated).**
    - names = { 'Darshit', 'Ragi', 'Aashna' }
    - names.add( 'Sharda' )
  + **If we want an immutable set, we should use a frozenset.**
    - names = frozenset({ 'Darshit', 'Ragi', 'Aashna' })
    - names.add( 'Sharda' ) 🡨 Error
  + **Concatenation does not work on sets.**
  + **Two sets can’t be concatenated using +.**
  + **Two sets can’t be merged to create a new set.**
  + **Conversion:** 
    - **A string/tuple/list can be converted into a set using the set ( ) conversion function.**
      * lst = [10, 20, 10, 30, 40 , 50 , 30 ]
      * s = set(lst)
      * print (s) 🡨 {40, 10, 50, 20, 30} 🡨 Duplicate entries will be eliminated.
  + **Aliasing:** 
    - **On assigning one set to another, both refer to the same set. Changing one changes the other.**
    - **Also known as shallow copy or aliasing.**
      * set1 = { 15, 8 , 6 }
      * set2 = set1 🡨 doesn’t copy set. set2 refers to the same set as set1.
      * print (set1 is set2) 🡨 True
  + **Cloning:** 
    - **This involves copying contents of one set into another.**
    - **After copying both refer to different sets, although both contain same values.**
    - **Changing one set doesn’t change another.**
    - **Also known as a deep copy.**
      * set1 = { 15, 8 , 6}
      * set2 = set1.copy()
      * print (set1 is set2) 🡨 False
  + **Searching:**
    - **An element can be searched in a set using the [in] membership operator.**
      * print (15 **in** set1) 🡨 True
  + **Identity:**
    - **Use [is] operator to check whether the two variables are referring to the same set.**
      * set1 = {10 , 20 , 30 , 40 , 50 }
      * set2 = {10 , 20 , 30 , 40 , 50 }
      * set3 = set1
      * print (set1 is set2, set1 is not set2, set1 is set3) 🡨 False, True, True
  + **Comparison:**
    - **We can compare contents of two sets.**
    - **Comparison is done item by item till there is a mismatch.**
      * A = { 1, 2, 3, 4 }
      * B = { 1, 2, 5 }
      * print ( A < B) 🡨 False , as A is not Superset of B.
  + **Emptiness:** 
    - **We can check if a set is empty using not operator.**
      * s = set ( )
      * if not s:
        + print (“Empty set.”)
      * print (bool(s)) 🡨 False
  + **Built-in functions on sets**
    - len(s) - max(s) - min(s)
    - sum(s) - any(s) - all(s)
  + **Set Methods**
    - **Any set is an object of type set.**
    - **Its methods can be accessed using the syntax set.method( )**
      * s = {72, 60, 58 , 52, 54, 56 }
      * t = { ‘a’, ‘b’, ‘c’ }
      * u = set ( )
      * s.add(74) # to add 74.
      * s.update(t) # adds elements of t to s.
      * s.remove(58) # deletes 58 from the set.
      * s.pop() #removes random element from a set
      * s.remove(50) # reports valueError as 50 is not present in the set.
      * s.discard(58) # removes 58 from the set.
      * s.discard(508) # Won’t raise an error, though 508 is not in set.
      * s.clear() # removes all elements.
    - **Following methods can be used on 2 sets to check the relationship between them:**
      * s = { 12, 15, 13, 23, 22 , 16, 17 }
      * t = { 13, 15, 23 }
      * print(s.**issuperset(t)** ) 🡨 Prints True
      * s >= t 🡨 Prints True
      * s > t 🡨 Prints True
      * {3, 5 ,4} > {5, 3, 4} 🡨 Prints False
      * print(s.**issubset(t)** ) 🡨 Prints False
      * s < = t 🡨 Prints False
      * s < t 🡨 Prints False
      * print(s.**isdisjoint(t)** ) 🡨 Prints False
        + If all elements of t are present in s, s is a superset of t and t is subset of s.
        + If intersection of two sets is null, the sets are called disjoint sets.
  + **Mathematical Set Operations**
    - Following union, intersection and different operations can be carried out on sets:
      * # sets
      * e = { 'Vijay', 'Sanjay', 'Ajay', 'Sujay', 'Dinesh' }
      * m = { 'Aaditya', 'Sanjay' }
      * # union - all people in both sets.
      * print (e | m )
      * z = e.union(m)
      * z = e.union(m, n)
      * z = e | m | n
      * # intersection - who are engineers and managers
      * print (e & m)
      * z = e.intersection(m)
      * z = e.intersection(m, n)
      * z = e & m & n
      * # difference - engineers who are not managers
      * print (e - m) or z = e - m
      * z = e.difference(m)
      * z = a – b – c (First a – b, then apply result – c ).
      * z = a.difference(b, c)
      * # difference - managers who are not engineers.
      * print (m - e)
      * # symmetric difference - managers who are not engineers
      * # and engineers who are not managers. Returns a set that contains all items from both set, but not the items that are present in both sets.
      * print (e ^ m)
      * z = x.symmetric\_difference(y)
  + **Updating Set Operations**
  + **Mathematical set operations can be extended to update an existing set.**
    - a |= b 🡨 Update a with the result of a | b.
    - a.update(b) 🡨 Update a with the result of a | b.
    - a.update(b, c) 🡨 Update a with the result of a | b | c.
    - a |= b | c 🡨 Update a with the result of a | b | c.
    - a &= b 🡨 update a with the result of a & b.
    - a.intersection\_update(b) 🡨 update a with the result of a & b.
    - a -= b 🡨 update a with the result of a – b.
    - a.differenece\_update(b) 🡨 update a with the result of a – b.
    - a ^= b 🡨 update a with the result of a ^ b.
    - x.symmetric\_difference\_update(y) 🡨 update x with the result of x ^ y.
  + **Set Varieties**
    - **Unlike a list and tuple, a set can’t contain nested set.**
      * s = { 1, 3, 5, { 7, 9} , 11 } 🡨 Error.
    - It is possible to unpack a set using the \* operator.
      * b = { 2, 4, 6, 8, 10 }
      * print (\*b) 🡨 prints 2, 4,6, 8, 10.
  + **Set Comprehension**
    - Set comprehensions offer an easy way of creating sets. it consists of braces containing an expression followed by a **for** clause, and zero or more **for** or **if** clauses.
    - **General Form:**
      * s = { expression **for** var **in** sequence [optional **for** and/or **if ] }**
    - **Examples:**
    - Create a set with squares of integers from 1 to 10.
      * squares = { x\*\*2 for x in range(1, 11) }
    - From a set , keep all numbers between 20 and 50.
      * b = { num for num in squares if num >= 20 and num <= 50 }
* **Write following programs considering sets in mind:**

1. Write a program that converts words present in a list into uppercase and stores them in a set.
2. Write a program to create a set containing 10 random numbers in the range of 15 to 45. Count how many of these numbers are less than 30. Delete all numbers that are greater than 35.
3. Create an empty set. Write a program that adds five new names to this set, modifies one existing name and deletes two names from it.
4. A set contains names which begin either with A or with B. Write a program to separate out the names into two sets, one containing names beginning with A and another with B.