Python Collections or Data structures (Arrays)

- There are four collection data types in the Python programming language:
- List is a collection which is ordered and changeable. Allows duplicate members.[]
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members. ()
- Set is a collection which is unordered and unindexed. No duplicate members.{}
- Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.{}

In [1]:
```
object = {1,2,1,1,1,2,2,2,2,4,5,5,5,5,6,6,6,6,7,7,7}
object
```

Out[1]:
```
{1, 2, 4, 5, 6, 7}
```

- storing multiple values and those can be multiple data types...

Set

- A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets.

In [3]:
```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
basket    # show that duplicates have been removed
```

Out[3]:
```
{'apple', 'banana', 'orange', 'pear'}
```

In [5]:
```
a_sets = {11,2,2,2,2,2,111,1,1,1,1,1,1,2,3,4,5,55,5,"Krishna","ram",6,6,7,7,8,9,10,'80',
type(a_sets)
```

Out[5]:
```
set
```

In [6]:
```
help(a_sets)
```

```
Help on set object:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
```

```
|  __reduce__(...)
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  add(...)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(...)
|      Remove all elements from this set.
|
|  copy(...)
|      Return a shallow copy of a set.
|
|  difference(...)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(...)
|      Remove all elements of another set from this set.
|
|  discard(...)
|      Remove an element from a set if it is a member.
|
|      If the element is not a member, do nothing.
|
|  intersection(...)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(...)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(...)
|      Return True if two sets have a null intersection.
|
|  issubset(...)
|      Report whether another set contains this set.
|
|  issuperset(...)
|      Report whether this set contains another set.
```

```
|
|  pop(...)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
|
|  remove(...)
|      Remove an element from a set; it must be a member.
|
|      If the element is not a member, raise a KeyError.
|
|  symmetric_difference(...)
|      Return the symmetric difference of two sets as a new set.
|
|      (i.e. all elements that are in exactly one of the sets.)
|
|  symmetric_difference_update(...)
|      Update a set with the symmetric difference of itself and another.
|
|  union(...)
|      Return the union of sets as a new set.
|
|      (i.e. all elements that are in either set.)
|
|  update(...)
|      Update a set with the union of itself and others.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

In [7]:
```
thisset = {"apple", "banana", "cherry"}
print(thisset)
type(thisset)
```

Out[7]:
```
{'cherry', 'apple', 'banana'}
set
```

Access Items

- You cannot access items in a set by referring to an index or a key.
- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

In [8]:
```
thisset = {"apple", "banana", "cherry","apple", "banana", "cherry","apple", "banana", "c
for x in thisset:
  print(x)
```

```
cherry
apple
banana
```

In [9]:
```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

In [10]:
```
# list - insert(index,value) ,extend and append
# set - add() - single value , update() for multiple values ([])
```

Add Items

- To add one item to a set use the add() method.
- To add more than one item to a set use the update() method.

In [11]:
```
thisset = {"apple", "banana", "cherry"}
thisset.add("range")
print(thisset)
```

```
{'cherry', 'range', 'apple', 'banana'}
```

In [12]:
```
help(thisset)
```

```
Help on set object:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
```

```
|  __reduce__(...)
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  add(...)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(...)
|      Remove all elements from this set.
|
|  copy(...)
|      Return a shallow copy of a set.
|
|  difference(...)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(...)
|      Remove all elements of another set from this set.
|
|  discard(...)
|      Remove an element from a set if it is a member.
|
|      If the element is not a member, do nothing.
|
|  intersection(...)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(...)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(...)
|      Return True if two sets have a null intersection.
|
|  issubset(...)
|      Report whether another set contains this set.
|
|  issuperset(...)
|      Report whether this set contains another set.
```

```
|
|  pop(...)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
|
|  remove(...)
|      Remove an element from a set; it must be a member.
|
|      If the element is not a member, raise a KeyError.
|
|  symmetric_difference(...)
|      Return the symmetric difference of two sets as a new set.
|
|      (i.e. all elements that are in exactly one of the sets.)
|
|  symmetric_difference_update(...)
|      Update a set with the symmetric difference of itself and another.
|
|  union(...)
|      Return the union of sets as a new set.
|
|      (i.e. all elements that are in either set.)
|
|  update(...)
|      Update a set with the union of itself and others.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

- Add multiple items to a set, using the **update()** method:

In [13]:
```
thisset = {"apple", "banana", "cherry"}
thisset.update(["abc","test1","test2"])
print(thisset)
```

```
{'cherry', 'abc', 'test1', 'banana', 'apple', 'test2'}
```

Get the Length of a Set

- To determine how many items a set has, use the **len()** method.

In [14]:
```
a_set ={1,2,3,4,5,6,7,8,9,10}
len(a_set)
```

Out[14]:
```
10
```

In [15]:
```
```

Out[15]:
```
10
```

In [16]:
```
thisset = {'mango', 'grapes', 'banana', 'orange', 'apple', 'cherry'}
print(len(thisset))
```

```
6
```

In [17]:
```
help(thisset)
```

```
Help on set object:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __or__(self, value, /)
 |      Return self|value.
 |
 |  __rand__(self, value, /)
 |      Return value&self.
 |
```

```
|  __reduce__(...)
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  add(...)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(...)
|      Remove all elements from this set.
|
|  copy(...)
|      Return a shallow copy of a set.
|
|  difference(...)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(...)
|      Remove all elements of another set from this set.
|
|  discard(...)
|      Remove an element from a set if it is a member.
|
|      If the element is not a member, do nothing.
|
|  intersection(...)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(...)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(...)
|      Return True if two sets have a null intersection.
|
|  issubset(...)
|      Report whether another set contains this set.
|
|  issuperset(...)
|      Report whether this set contains another set.
```

```
|
|  pop(...)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
|
|  remove(...)
|      Remove an element from a set; it must be a member.
|
|      If the element is not a member, raise a KeyError.
|
|  symmetric_difference(...)
|      Return the symmetric difference of two sets as a new set.
|
|      (i.e. all elements that are in exactly one of the sets.)
|
|  symmetric_difference_update(...)
|      Update a set with the symmetric difference of itself and another.
|
|  union(...)
|      Return the union of sets as a new set.
|
|      (i.e. all elements that are in either set.)
|
|  update(...)
|      Update a set with the union of itself and others.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

In [19]:
```
# list remove values - remove (),pop(),del list[] ,clear()
```

Remove Item

- To remove an item in a set, use the **remove()**, or the **discard()** method.

In [20]:
```
thisset = {"apple", "banana", "cherry","kiwi"}
thisset.remove("bananass")
print(thisset)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [20], in <cell line: 2>()
      1 thisset = {"apple", "banana", "cherry","kiwi"}
----> 2 thisset.remove("bananass")
      3 print(thisset)

KeyError: 'bananass'
```

In [ ]:
```
thisset = {"apple", "banana", "cherry","kiwi"}
thisset.discard("bananas")
print(thisset)
```

In [21]:
```
list_A = [1,2,3,4]
list_A.append(5)
list_A.append(6)
print('before deletion ',list_A)
list_A.pop()
print('after deletion ',list_A)
list_A.pop()
print('after deletion ',list_A)
```

```
before deletion  [1, 2, 3, 4, 5, 6]
after deletion  [1, 2, 3, 4, 5]
after deletion  [1, 2, 3, 4]
```

- You can also use the **pop()**, method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed.
- The return value of the pop() method is the removed item.

In [22]:
```
thisset = {"apple", "banana", "cherry","kiwi"}
x = thisset.pop()
print(x)
print(thisset)
```

```
cherry
{'kiwi', 'apple', 'banana'}
```

In [23]:
```
# indexes it will follow range ()
# String,list,tuple,dict - with updatable.
# table data. rows (rowid)
# set operators ( union,intersection,minus(subtract))
```

In [24]:
```
x = thisset.pop()
print(x)
```

```
kiwi
```

- The **clear()** method empties the set:

In [25]:
```
thisset = {"apple", "banana", "cherry"}
print(thisset)
thisset.clear()
thisset
# {}
```

```
{'cherry', 'apple', 'banana'}
```
Out[25]:
```
set()
```

- The `del` keyword will delete the set completely:

In [26]:
```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [26], in <cell line: 3>()
      1 thisset = {"apple", "banana", "cherry"}
      2 del thisset
----> 3 print(thisset)

NameError: name 'thisset' is not defined
```

In [27]:
```
help(set3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [27], in <cell line: 1>()
----> 1 help(set3)

NameError: name 'set3' is not defined
```

Join Two Sets

- There are several ways to join two or more sets in Python.
- You can use the union() method that returns a new set containing all items from both sets.
- update() method that inserts all the items from one set into another

In [28]:
```
set1 = {"a", "b" , "c",1,2,3,4,5,6,66,99}
set1
set2 = {1, 2, 3,"c",23,45,"test11"}
set2.intersection(set1)  # set1 - set2  # set2- set1  # difference,subtract,minus
```

Out[28]:
```
{1, 2, 3, 'c'}
```

In [31]:
```
set1 = {"a", "b" , "c",1,2,3,4,5,6,66,99}
set2 = {1, 2, 3,"c",23,45,"test11"}
set3 = set2.union(set1)
set2.update(set1)
print('union set : ',set3)
print('Updating existing set : ',set2)
```

```
union set :  {1, 2, 3, 'test11', 4, 'c', 5, 6, 66, 'b', 'a', 23, 99, 45}
Updating existing set :  {1, 2, 3, 'test11', 4, 'c', 5, 6, 66, 'b', 'a', 23, 99, 45}
```

In [32]:
```
# Database set operators
# Dataset ( similar to table)
# set operators
# 1) union - merge two sets  and its remove duplicates
# 2) intersection - getting common matching items from two sets
# 3) difference (subtract or minus) - getting A-B or Set1- Set2  # its equal to subtract
# unionall will return duplicate.
```

In [33]:
```
#same data type.
```

In [34]:
```
set1 ={1,2,3,4,5,6}
set2 = {3,4,5,6,7,8}
set2.intersection(set1)
# subtract , minus
```

Out[34]:
```
{3, 4, 5, 6}
```

- The update() method inserts the items in set2 into set1

In [35]:
```
set1 = {"a", "b" , "c",1,2,3,}
set2 = {1, 2, 3,4,5,"b"}
set1.intersection(set2)
print(set1)
```

```
{1, 2, 3, 'a', 'c', 'b'}
```

In [36]:
```
# A Set of names
names = {"Ravi", "Raj", "Ram"}

# copying using the copy() method
names2 = names.copy()
print(names)
print(names2)
```

```
{'Raj', 'Ram', 'Ravi'}
{'Raj', 'Ram', 'Ravi'}
```

In [37]:
```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'f', 'g'}
print(A.difference(B)) # Equivalent to A-B ( it is equal to minus operator)
print(B.difference(A)) # Equivalent to B-A ( it is equal to minus operator)
```

```
{'b', 'a', 'd'}
{'f', 'g'}
```

Note: when we are doing data set set operators.

A Difference B and B Difference A will get different result set but A union B, B union A and A intersection B and B intersection A will get same results.

In [38]:
```
A = {'a', 'b', 'c', 'd','e'}
B = {'c','d','e', 'f', 'g'}
# Equivalent to common matching values
print(A.intersection(B))
```

```
{'e', 'c', 'd'}
```