SETS

A set is a collection which is unordered, unchangeable*, and unindexed.

In [180…

```
x = set() # Initialize an empty set and assign it to the variable 'x'
print(x)
print(type(x))

print("////////////////////")
n = set([0, 1, 2, 3, 4])
print(n)
print(type(n))
print("////////////////////")
a = {1, 2, 3, 'foo', 'bar'}
print(a)
print(type(a))
print("////////////////////")
a = set((1, 2, 3, 'foo', 'bar',True,False)) #Here True and 1 are treated as same element
print(a) #tuple will not allow the duplicates , so True is not printing
print(type(a))
```

```
set()
<class 'set'>
////////////////////
{0, 1, 2, 3, 4}
<class 'set'>
////////////////////
{1, 2, 3, 'foo', 'bar'}
<class 'set'>
////////////////////
{False, 1, 2, 3, 'foo', 'bar'}
<class 'set'>
```

In [182…

```
#loop or accessing the set elements
num_set = set([0, 1, 2, 3, 4, 5])
for n in num_set:
    print(n, end=' ')
print("\n")
char_set = set("Digicomm semiconductor")
for val in char_set:
    print(val, end=' ')
```

```
0 1 2 3 4 5

e o i n c m r g s D   u t d
```

In [184…

```
#for loop with index
char_set = set("Digicomm semiconductor")
print(char_set)
char_set_list =list(char_set)
for val in range(len(char_set_list)):
    print(char_set_list[val],end="")
```

```
{'e', 'o', 'i', 'n', 'c', 'm', 'r', 'g', 's', 'D', ' ', 'u', 't', 'd'}
eoincmrgsD utd
```

In [188…

```
#Adding elements to the sets using add and update method
color_set = set()
print(color_set)
color_set.add("Red") #To add one item to a set use the add() method.
print(color_set)
# Add multiple elements "Blue" and "Green" to the 'color_set' using the 'update' method:
color_set.update(["Blue", "Green"])
print(color_set)
# Add items from another set into the current set, use the update() method.
color_set.update({"White","yellow"})
print(color_set)
```

```
set()
{'Red'}
{'Green', 'Blue', 'Red'}
{'yellow', 'Blue', 'Red', 'White', 'Green'}
```

In [62]:

```
color_set={'yellow', 'Blue', 'Red', 'White', 'Green'}
print(color_set)
color_set.remove("yellow")
print(color_set)
```

```
{'yellow', 'Blue', 'Red', 'White', 'Green'}
{'Blue', 'Red', 'White', 'Green'}
```

In [66]:

```
color_set={'Blue', 'Red', 'White', 'Green'}
color_set.remove("yellow")
print(color_set) #If the item to remove does not exist, remove() will raise an error.
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[66], line 2
      1 color_set={'Blue', 'Red', 'White', 'Green'}
----> 2 color_set.remove("yellow")
      3 print(color_set)

KeyError: 'yellow'
```

In [70]:

```
color_set={'Blue', 'Red', 'White', 'Green'}
color_set.discard("yellow")
print(color_set)
color_set.discard("White")
print(color_set)
```

```
{'Green', 'Blue', 'Red', 'White'}
{'Green', 'Blue', 'Red'}
```

In [ ]:

```
#You can also use the pop() method to remove an item, but this method will remove a random item,
#so you cannot be sure what item that gets removed.
#The return value of the pop() method is the removed item.
```

In [190…

```
color_set={'Blue', 'Red', 'White', 'Green'}
x=color_set.pop()
print(x)
print(color_set)
```

```
Green
{'Blue', 'Red', 'White'}
```

In [74]:

```
#remove items from the sets:
def remove(initial_set):
    while initial_set:
        initial_set.pop()
        print(initial_set)

initial_set=set([2,4,6,3,8,0,5,1])
remove(initial_set)
```

```
{1, 2, 3, 4, 5, 6, 8}
{2, 3, 4, 5, 6, 8}
{3, 4, 5, 6, 8}
{4, 5, 6, 8}
{5, 6, 8}
{6, 8}
{8}
set()
```

In [76]:

```
color_set={'Blue', 'Red', 'White', 'Green'}
color_set.clear()
print(color_set)
```

```
set()
```

In [82]:

```
color_set={'Blue', 'Red', 'White', 'Green'}
print(color_set)
del color_set
print(color_set)
```

```
{'Green', 'Blue', 'Red', 'White'}
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[82], line 4
      2 print(color_set)
      3 del color_set
----> 4 print(color_set)

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

In [212…

```
a={2,4,6,8,10}
b={1,3,5,7,9}
c=b.union(a)
d=a.union(b)
e=a|b
print(a)
print(b)
print(c)
print(d)
print(e)
#a=append(b)
#[2,4,6,8,10,1,3,5,7,9]
```

```
{2, 4, 6, 8, 10}
{1, 3, 5, 7, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [ ]:

```
'''
union               : Return a set that contains all items from both sets, duplicates are excluded
intersection        : Return a set that contains the items that exist in both set
difference          : Return a set that contains the items that only exist in set x, and not in set y
symmetric_difference : Return a set that contains all items from both sets, except items that are present in both sets
'''
```

In [196…

```
# (union,|) (intersection,&) (difference,-) (symmetric_difference,^)
a={"a","c","e"}
b={"b","d","f"}
m={1,2,4,5,6}
n={11,22,33,44}
p=(12,13,14)
q={1,2,"a","c"}
c=a.union(b)
print(f" union of {a} and {b} is {c}")
d=b.union(a)
print(f" union of {b} and {a} is {d}")
e=a|b
print(f" union of {a} and {b} is {e}")
s=a.union(b,m,n)
print(f" union of {a},{b},{m},{n} is {s}")
g=a|b|m|n
print(f" union with operator : {a},{b},{m},{n} is {g}")
h=a.union(b,m,n,p)
print(f" union with tuple : {a},{b},{m},{n} and {p} is {h}")
#k=a|b|m|n|p #unsupported operand type(s) for |: 'set' and 'tuple'
#print(k)
x=a.intersection(q)
print(f" intersection of {a} and {q} is {x}")
y=a&q #only sets with sets not support with other data types
print(f" intersection with operator: {a} and {q} is {y}")
x=a.difference(q)
print(f"difference b/w {a} and {q} is {x}")
x=q.difference(a)
print(f"difference b/w {q} and {a} is {x}")
x=a-q
print(f"sub b/w {a} and {q} is {x}")
x=a.symmetric_difference(q)
print(f"symmetric_difference b/w {a} and {q} is {x}")
x=q.symmetric_difference(a)
print(f"symmetric_difference b/w {q} and {a} is {x}")
x=q^a
print(f"symmetric_difference b/w {q} and {a} is {x}")

#a=a-b
print("////////////////////")
print(f"Before difference_update {a} and {q}")
a.difference_update(q)
print(f"After difference_update {a} and {q}")
print("////////////////////")
a={"a","c","e"}
print(f"Before intersection_update {a} and {q}")
a.intersection_update(q)
print(f"After intersection_update {a} and {q}")
```

```
 union of {'c', 'e', 'a'} and {'b', 'd', 'f'} is {'e', 'f', 'a', 'b', 'c', 'd'}
 union of {'b', 'd', 'f'} and {'c', 'e', 'a'} is {'e', 'a', 'f', 'b', 'c', 'd'}
 union of {'c', 'e', 'a'} and {'b', 'd', 'f'} is {'e', 'f', 'a', 'b', 'c', 'd'}
 union of {'c', 'e', 'a'},{'b', 'd', 'f'},{1, 2, 4, 5, 6},{33, 11, 44, 22} is {'e', 1, 'a', 2, 4, 5, 6, 33, 'c', 11, 44, 22, 'f', 'b', 'd'}
 union with operator : {'c', 'e', 'a'},{'b', 'd', 'f'},{1, 2, 4, 5, 6},{33, 11, 44, 22} is {'e', 1, 'a', 2, 4, 5, 6, 33, 'c', 11, 44, 22, 'f', 'b', 'd'}
 union with tuple : {'c', 'e', 'a'},{'b', 'd', 'f'},{1, 2, 4, 5, 6},{33, 11, 44, 22} and (12, 13, 14) is {'e', 1, 'a', 2, 4, 5, 6, 33, 'c', 11, 44, 12, 13, 14, 22, 'f', 'b', 'd'}
 intersection of {'c', 'e', 'a'} and {1, 2, 'c', 'a'} is {'c', 'a'}
 intersection with operator: {'c', 'e', 'a'} and {1, 2, 'c', 'a'} is {'c', 'a'}
difference b/w {'c', 'e', 'a'} and {1, 2, 'c', 'a'} is {'e'}
difference b/w {1, 2, 'c', 'a'} and {'c', 'e', 'a'} is {1, 2}
sub b/w {'c', 'e', 'a'} and {1, 2, 'c', 'a'} is {'e'}
symmetric_difference b/w {'c', 'e', 'a'} and {1, 2, 'c', 'a'} is {1, 2, 'e'}
symmetric_difference b/w {1, 2, 'c', 'a'} and {'c', 'e', 'a'} is {'e', 2, 1}
symmetric_difference b/w {1, 2, 'c', 'a'} and {'c', 'e', 'a'} is {'e', 2, 1}
////////////////////
Before difference_update {'c', 'e', 'a'} and {1, 2, 'c', 'a'}
After difference_update {'e'} and {1, 2, 'c', 'a'}
////////////////////
Before intersection_update {'c', 'e', 'a'} and {1, 2, 'c', 'a'}
After intersection_update {'c', 'a'} and {1, 2, 'c', 'a'}
```

In [156…

```
a={"a","c","e"}
b=a.copy()
print(b)
```

```
{'c', 'e', 'a'}
```

In [7]:

```
#max and min value in the sets:
def maxm(sets):
    return(max(sets))
def minm(sets):
    return(min(sets))
sets=set([2,4,7,3,1,8,0])
print(maxm(sets))
print(minm(sets))
```

```
8
0
```

In [11]:

```
def intersection_of_sets(arr1,arr2,arr3):
    s1=set(arr1)
    s2=set(arr2)
    s3=set(arr3)
    set1=s1.intersection(s2)
    result_set=set1.intersection(s3)
    final_list=list(result_set)
    print(final_list)

arr1=[1,5,10,20,40,80,100]
arr2=[6,7,20,80,100]
arr3=[3,4,15,20,30,70,80,120]
intersection_of_sets(arr1,arr2,arr3)
```

```
[80, 20]
```

In [15]:

```
#accept the string which contain all vowels:
def check(string):
    string=string.lower()
    vowels=set("aeiou")
    s=set({})
    for char in string:
        if char in vowels:
            s.add(char)
        else:
            pass
    if len(s)==len(vowels):
        print("accepted")
    else:
        print("not accepted")
string="SEEqoUial"
check(string)
```

```
accepted
```

In [198…

```
students_english = {"Alice", "Bob", "Charlie", "David"}
students_math = {"Charlie", "Eve", "Frank", "Alice"}
students_science = {"George", "Alice", "Frank", "David"}

# Students taking all three subjects
all_three_subjects = students_english & students_math & students_science

# Students taking at least one subject
at_least_one_subject = students_english | students_math | students_science

# Students taking only one subject
only_english = students_english - (students_math | students_science)
only_math = students_math - (students_english | students_science)
only_science = students_science - (students_english | students_math)

print("All three subjects:", all_three_subjects)
print("At least one subject:", at_least_one_subject)
print("Only English:", only_english)
print("Only Math:", only_math)
print("Only Science:", only_science)
```

```
All three subjects: {'Alice'}
At least one subject: {'Alice', 'Frank', 'Charlie', 'David', 'Eve', 'George', 'Bob'}
Only English: {'Bob'}
Only Math: {'Eve'}
Only Science: {'George'}
```

In [216…

```
yoga_class = {"Mike", "Sara", "Tom","Jerry"}
spinning_class = {"Tom", "Jerry", "Kate","Jerry"}
pilates_class = {"Sara", "Mike", "Jerry"}

# People signed up for all classes
signed_up_all_classes = yoga_class & spinning_class & pilates_class

# People signed up for at least one class
signed_up_any_class = yoga_class | spinning_class | pilates_class

# People signed up for exactly two classes
signed_up_two_classes = (yoga_class & spinning_class) |(spinning_class & pilates_class) | (yoga_class & pilates_class)
signed_up_two_classes -= signed_up_all_classes

print("Signed up for all classes:", signed_up_all_classes)
print("Signed up for any class:", signed_up_any_class)
print("Signed up for exactly two classes:", signed_up_two_classes)
```

```
Signed up for all classes: {'Jerry'}
Signed up for any class: {'Jerry', 'Mike', 'Kate', 'Tom', 'Sara'}
Signed up for exactly two classes: {'Mike', 'Tom', 'Sara'}
```

In [218…

```
event_A = {"John", "Jane", "Doe", "Alice"}
event_B = {"Jane", "Alice", "Tom", "Jerry"}
event_C = {"Tom", "Jerry", "Doe", "Harry"}

# Attendees who attended only one event
only_event_A = event_A.difference(event_B.union(event_C))
only_event_B = event_B.difference(event_A.union(event_C))
only_event_C = event_C.difference(event_A.union(event_B))

# Attendees who attended at least two events
at_least_two_events = (event_A.intersection(event_B)).union(
                      event_B.intersection(event_C)).union(
                      event_A.intersection(event_C))

print("Only attended Event A:", only_event_A)
print("Only attended Event B:", only_event_B)
print("Only attended Event C:", only_event_C)
print("Attended at least two events:", at_least_two_events)
```

```
Only attended Event A: {'John'}
Only attended Event B: set()
Only attended Event C: {'Harry'}
Attended at least two events: {'Alice', 'Jerry', 'Tom', 'Doe', 'Jane'}
```

In [170…

```
def find_combinations_of_three(nums, target_val):
    nums = list(set(nums))
    result = set()

    for i in range(len(nums)):
        for j in range(i+1, len(nums)):
            complement = target_val - nums[i] - nums[j]

            # Check if the 'complement' exists in the remaining part of the list 'nums'.
            if complement in nums[:i] + nums[j+1:]:
                result.add(tuple(sorted((nums[i], nums[j], complement))))

    return list(result)

# Define a list of numbers 'nums' and a 'target_val' for testing.
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9]
target_val = 12
print("Combine three numbers whose sum equals the target number:")
print(find_combinations_of_three(nums, target_val))

target_val = 17
print("Combine three numbers whose sum equals the target number:")
print(find_combinations_of_three(nums, target_val))
```

```
Combine three numbers whose sum equals the target number:
[(1, 3, 8), (1, 4, 7), (1, 2, 9), (1, 5, 6), (3, 4, 5), (2, 3, 7), (2, 4, 6)]
Combine three numbers whose sum equals the target number:
[(4, 5, 8), (2, 6, 9), (3, 5, 9), (2, 7, 8), (4, 6, 7), (3, 6, 8), (1, 7, 9)]
```

In [166]:
```
#Write a Python program to find the third largest number from a given list of numbers.Use the Python set data type
```

In [ ]:
```
list-set (1,3,5,7)
set-list (sort,index)
```