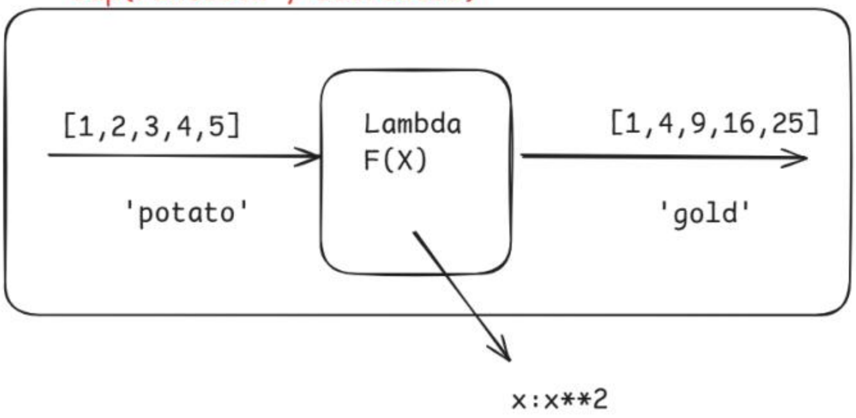
CN-Python-Lec06-Sets and Dictionary in Python

Sets and Dictionary in Python

🎯 Session Objectives

- 🔒 Understand what tuples are.
- 🛠️ Understand common methods and operations associated with tuples.
- 📚 Understand what sets are.
- ⚙️ Understand common methods and operations associated with sets.
- ⚖️ Understand the comparison between lists, tuples and sets.
- 🔑 Understand what dictionaries are.
- 🛠️ Understand common methods and operations associated with dictionaries.
- 🤝 Understand the comparison between lists, tuples, sets and dictionaries.

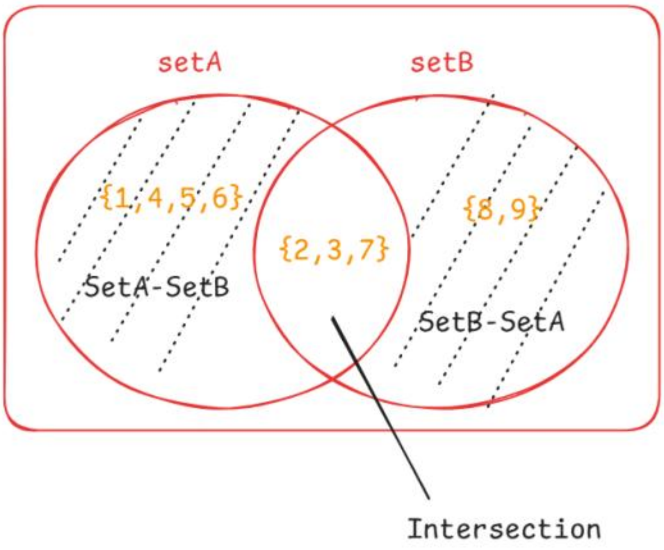
Map(Function , Iterables)



Functions<>

DRY - Don't Repeat Yourself

Set

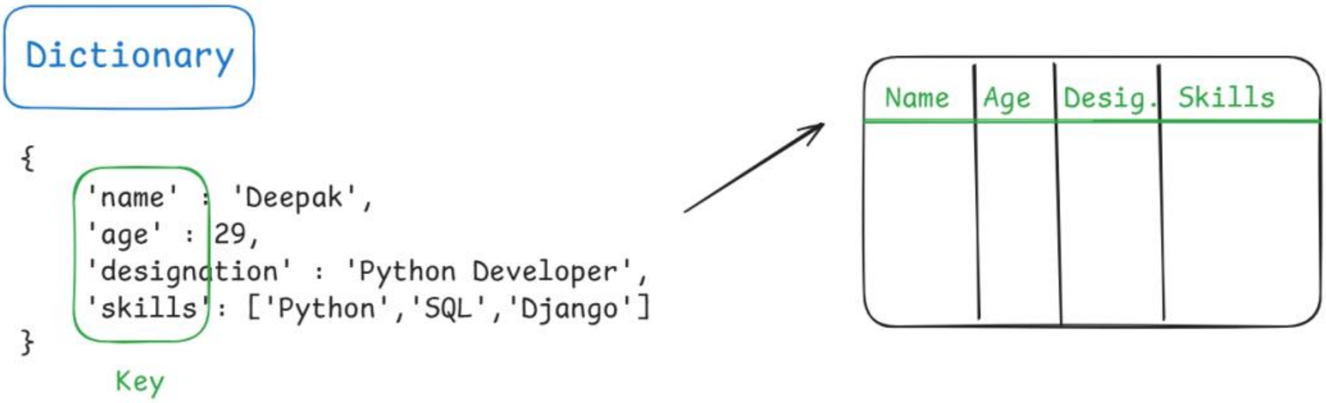


setA = {1,2,3,4,4,5,6,7,5}
setB = {2,2,3,3,7,7,8,8,9}

SetA U SetB = {1,2,3,4,5,6,7,8,9}

SetA ^ SetB = {1,4,5,6,8,9}

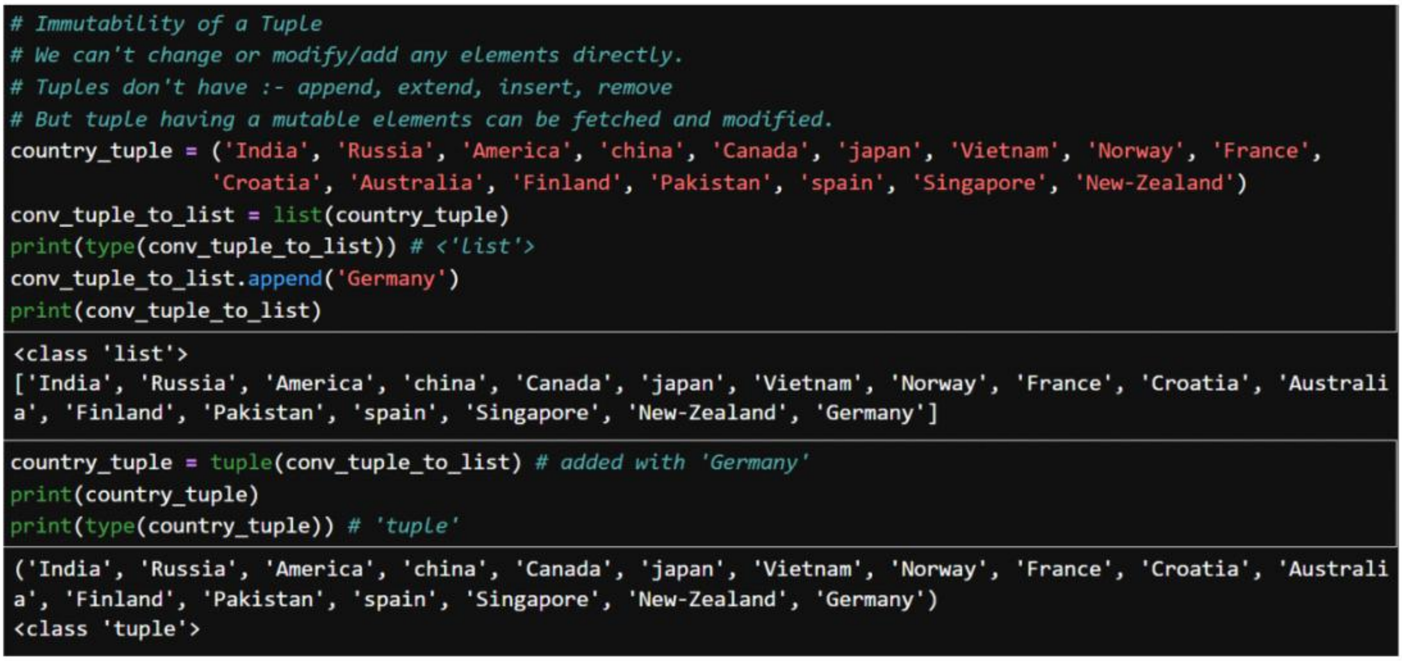
Symmetric Difference



Dictionary

```
{
    'name'  : 'Deepak',
    'age'  : 29,
    'designation' : 'Python Developer',
    'skills' : ['Python','SQL','Django']
}
```

Name	Age	Desig.	Skills

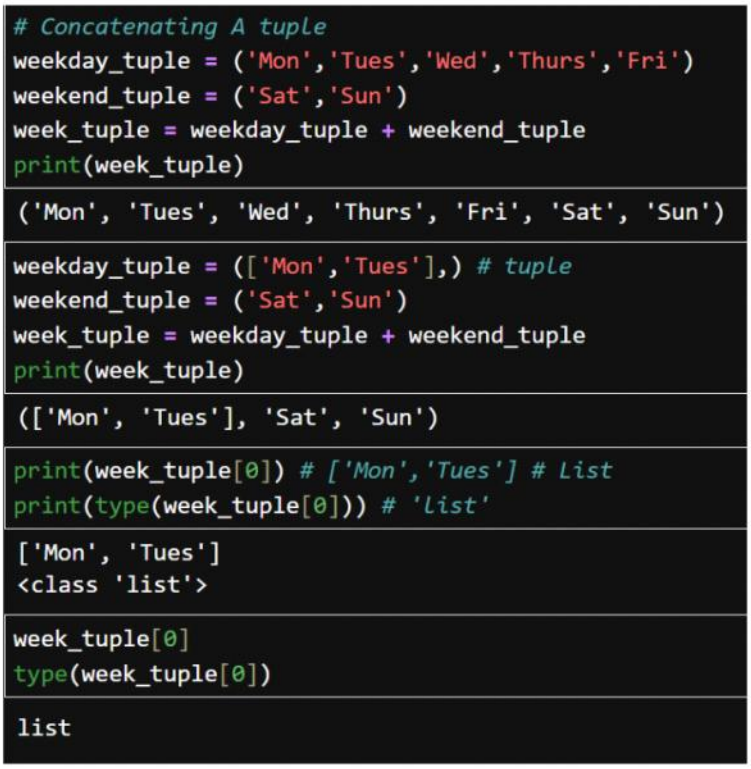


```python
# Immutability of a Tuple
# We can't change or modify/add any elements directly.
# Tuples don't have :- append, extend, insert, remove
# But tuple having a mutable elements can be fetched and modified.
country_tuple = ('India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France',
                 'Croatia', 'Australia', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand')
conv_tuple_to_list = list(country_tuple)
print(type(conv_tuple_to_list)) # <'list'>
conv_tuple_to_list.append('Germany')
print(conv_tuple_to_list)
```

```
<class 'list'>
['India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France', 'Croatia', 'Australi
a', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand', 'Germany']
```

```python
country_tuple = tuple(conv_tuple_to_list) # added with 'Germany'
print(country_tuple)
print(type(country_tuple)) # 'tuple'
```

```
('India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France', 'Croatia', 'Australi
a', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand', 'Germany')
<class 'tuple'>
```



```python
# Concatenating A tuple
weekday_tuple = ('Mon','Tues','Wed','Thurs','Fri')
weekend_tuple = ('Sat','Sun')
week_tuple = weekday_tuple + weekend_tuple
print(week_tuple)
```

```
('Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun')
```

```python
weekday_tuple = (['Mon','Tues'],) # tuple
weekend_tuple = ('Sat','Sun')
week_tuple = weekday_tuple + weekend_tuple
print(week_tuple)
```

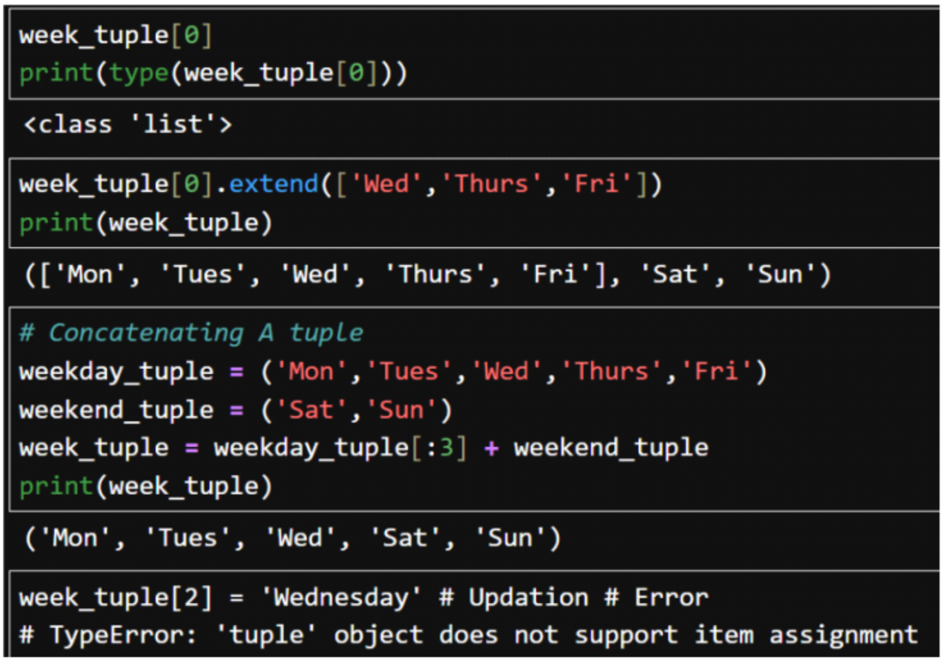
```
(['Mon', 'Tues'], 'Sat', 'Sun')
```

```python
print(week_tuple[0]) # ['Mon','Tues'] # List
print(type(week_tuple[0])) # 'list'
```

```
['Mon', 'Tues']
<class 'list'>
```

```python
week_tuple[0]
type(week_tuple[0])
```

```
list
```



```
week_tuple[0]
print(type(week_tuple[0]))
```

```
<class 'list'>
```

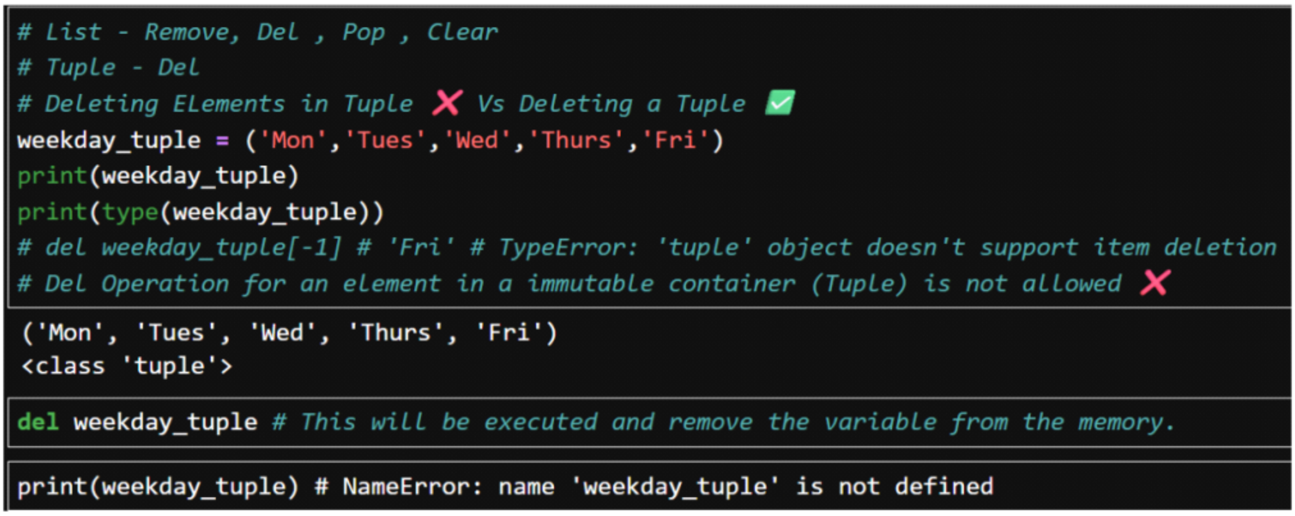
```
week_tuple[0].extend(['Wed','Thurs','Fri'])
print(week_tuple)
```

```
(['Mon', 'Tues', 'Wed', 'Thurs', 'Fri'], 'Sat', 'Sun')
```

```
# Concatenating A tuple
weekday_tuple = ('Mon','Tues','Wed','Thurs','Fri')
weekend_tuple = ('Sat','Sun')
week_tuple = weekday_tuple[:3] + weekend_tuple
print(week_tuple)
```

```
('Mon', 'Tues', 'Wed', 'Sat', 'Sun')
```

```
week_tuple[2] = 'Wednesday' # Updation # Error
# TypeError: 'tuple' object does not support item assignment
```

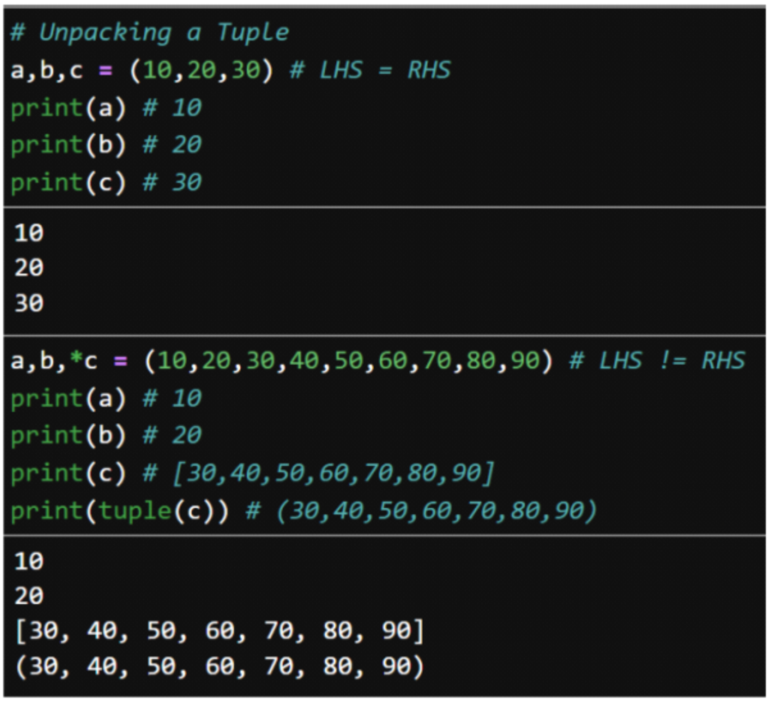


```
# List - Remove, Del , Pop , Clear
# Tuple - Del
# Deleting ELements in Tuple ❌ Vs Deleting a Tuple ✅
weekday_tuple = ('Mon','Tues','Wed','Thurs','Fri')
print(weekday_tuple)
print(type(weekday_tuple))
# del weekday_tuple[-1] # 'Fri' # TypeError: 'tuple' object doesn't support item deletion
# Del Operation for an element in a immutable container (Tuple) is not allowed ❌
```

```
('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')
<class 'tuple'>
```

```
del weekday_tuple # This will be executed and remove the variable from the memory.
```

```
print(weekday_tuple) # NameError: name 'weekday_tuple' is not defined
```

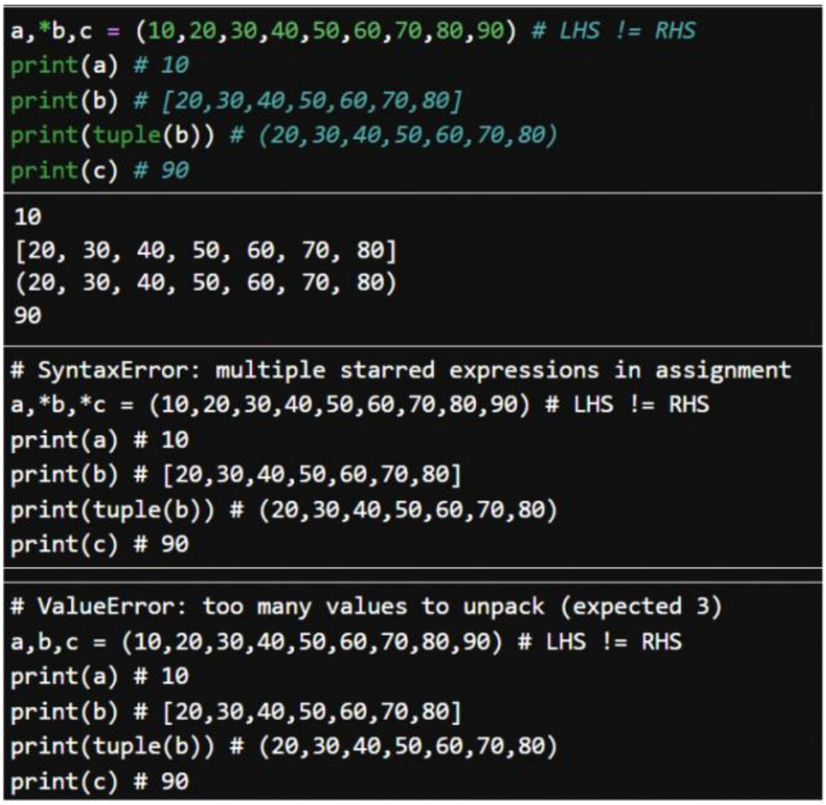


```
# Unpacking a Tuple
a,b,c = (10,20,30) # LHS = RHS
print(a) # 10
print(b) # 20
print(c) # 30
```

```
10
20
30
```

```
a,b,*c = (10,20,30,40,50,60,70,80,90) # LHS != RHS
print(a) # 10
print(b) # 20
print(c) # [30,40,50,60,70,80,90]
print(tuple(c)) # (30,40,50,60,70,80,90)
```

```
10
20
[30, 40, 50, 60, 70, 80, 90]
(30, 40, 50, 60, 70, 80, 90)
```

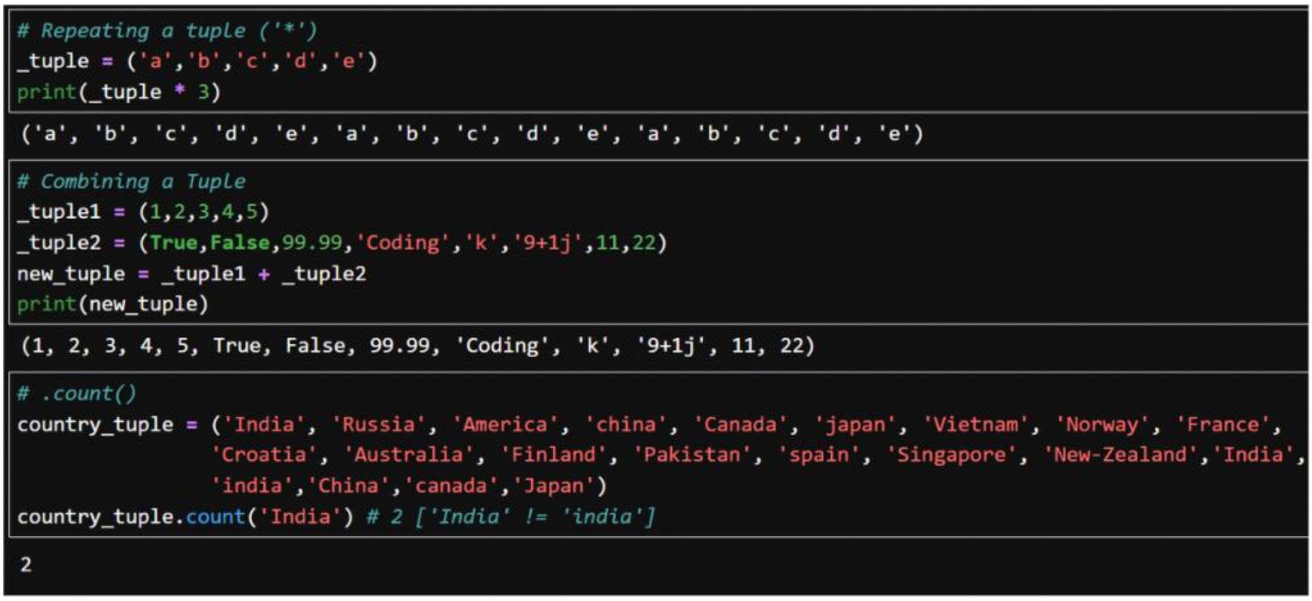


```
a,*b,c = (10,20,30,40,50,60,70,80,90) # LHS != RHS
print(a) # 10
print(b) # [20,30,40,50,60,70,80]
print(tuple(b)) # (20,30,40,50,60,70,80)
print(c) # 90
```

```
10
[20, 30, 40, 50, 60, 70, 80]
(20, 30, 40, 50, 60, 70, 80)
90
```

```
# SyntaxError: multiple starred expressions in assignment
a,*b,*c = (10,20,30,40,50,60,70,80,90) # LHS != RHS
print(a) # 10
print(b) # [20,30,40,50,60,70,80]
print(tuple(b)) # (20,30,40,50,60,70,80)
print(c) # 90
```

```
# ValueError: too many values to unpack (expected 3)
a,b,c = (10,20,30,40,50,60,70,80,90) # LHS != RHS
print(a) # 10
print(b) # [20,30,40,50,60,70,80]
print(tuple(b)) # (20,30,40,50,60,70,80)
print(c) # 90
```



```
# Repeating a tuple ('*')
_tuple = ('a','b','c','d','e')
print(_tuple * 3)
```

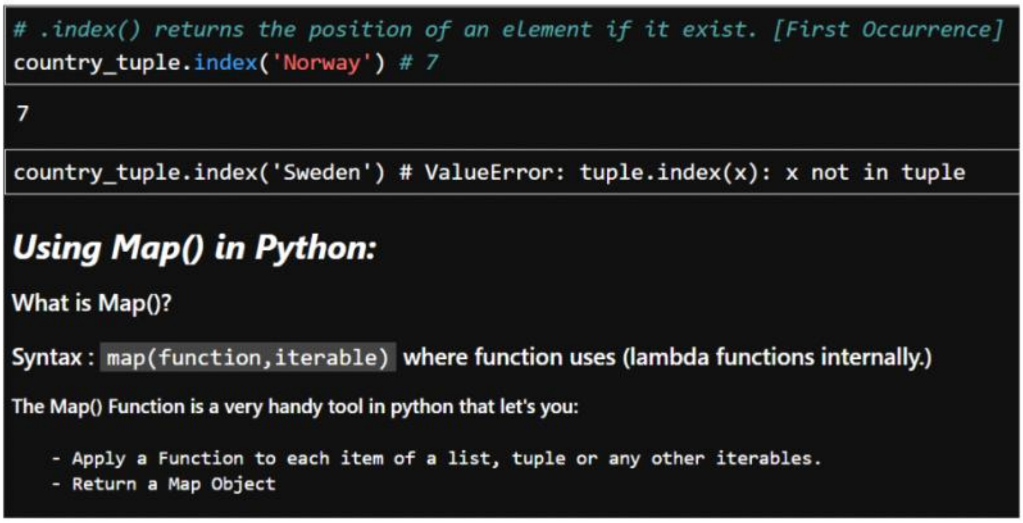
```
('a', 'b', 'c', 'd', 'e', 'a', 'b', 'c', 'd', 'e', 'a', 'b', 'c', 'd', 'e')
```

```
# Combining a Tuple
_tuple1 = (1,2,3,4,5)
_tuple2 = (True,False,99.99,'Coding','k','9+1j',11,22)
new_tuple = _tuple1 + _tuple2
print(new_tuple)
```

```
(1, 2, 3, 4, 5, True, False, 99.99, 'Coding', 'k', '9+1j', 11, 22)
```

```
# .count()
country_tuple = ('India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France',
                 'Croatia', 'Australia', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand','India',
                 'india','China','canada','Japan')
country_tuple.count('India') # 2 ['India' != 'india']
```

```
2
```



```
# .index() returns the position of an element if it exist. [First Occurrence]
country_tuple.index('Norway') # 7
```

```
7
```

```
country_tuple.index('Sweden') # ValueError: tuple.index(x): x not in tuple
```

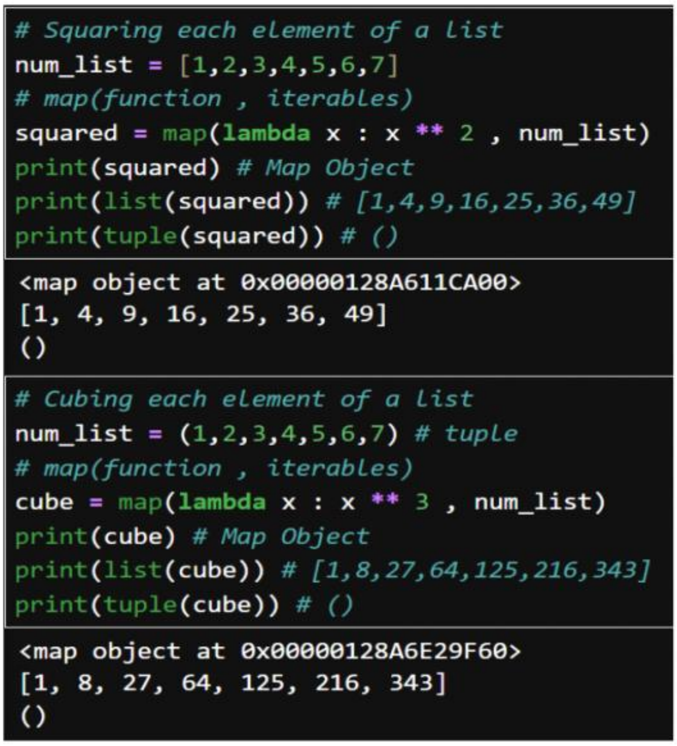
Using Map() in Python:

What is Map()?

Syntax : `map(function,iterable)` **where function uses (lambda functions internally.)**

The Map() Function is a very handy tool in python that let's you:

```
- Apply a Function to each item of a list, tuple or any other iterables.
- Return a Map Object
```

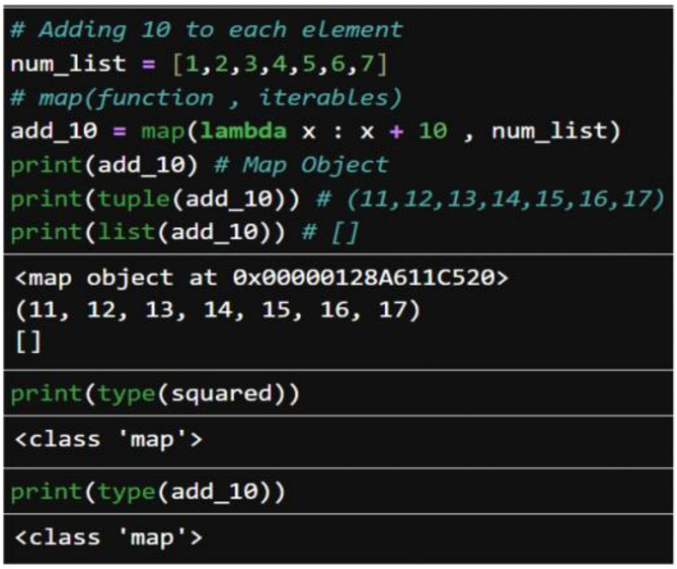


```
# Squaring each element of a list
num_list = [1,2,3,4,5,6,7]
# map(function , iterables)
squared = map(lambda x : x ** 2 , num_list)
print(squared) # Map Object
print(list(squared)) # [1,4,9,16,25,36,49]
print(tuple(squared)) # ()
```

```
<map object at 0x00000128A611CA00>
[1, 4, 9, 16, 25, 36, 49]
()
```

```
# Cubing each element of a list
num_list = (1,2,3,4,5,6,7) # tuple
# map(function , iterables)
cube = map(lambda x : x ** 3 , num_list)
print(cube) # Map Object
print(list(cube)) # [1,8,27,64,125,216,343]
print(tuple(cube)) # ()
```

```
<map object at 0x00000128A6E29F60>
[1, 8, 27, 64, 125, 216, 343]
()
```



```
# Adding 10 to each element
num_list = [1,2,3,4,5,6,7]
# map(function , iterables)
add_10 = map(lambda x : x + 10 , num_list)
print(add_10) # Map Object
print(tuple(add_10)) # (11,12,13,14,15,16,17)
print(list(add_10)) # []
```

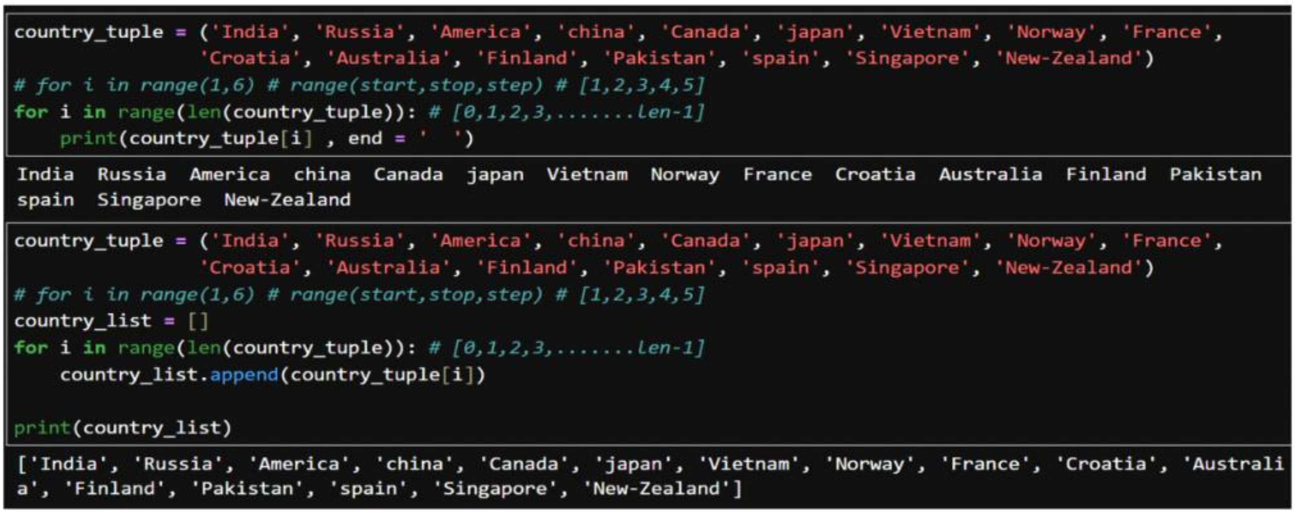
```
<map object at 0x00000128A611C520>
(11, 12, 13, 14, 15, 16, 17)
[]
```

```
print(type(squared))
```

```
<class 'map'>
```

```
print(type(add_10))
```

```
<class 'map'>
```



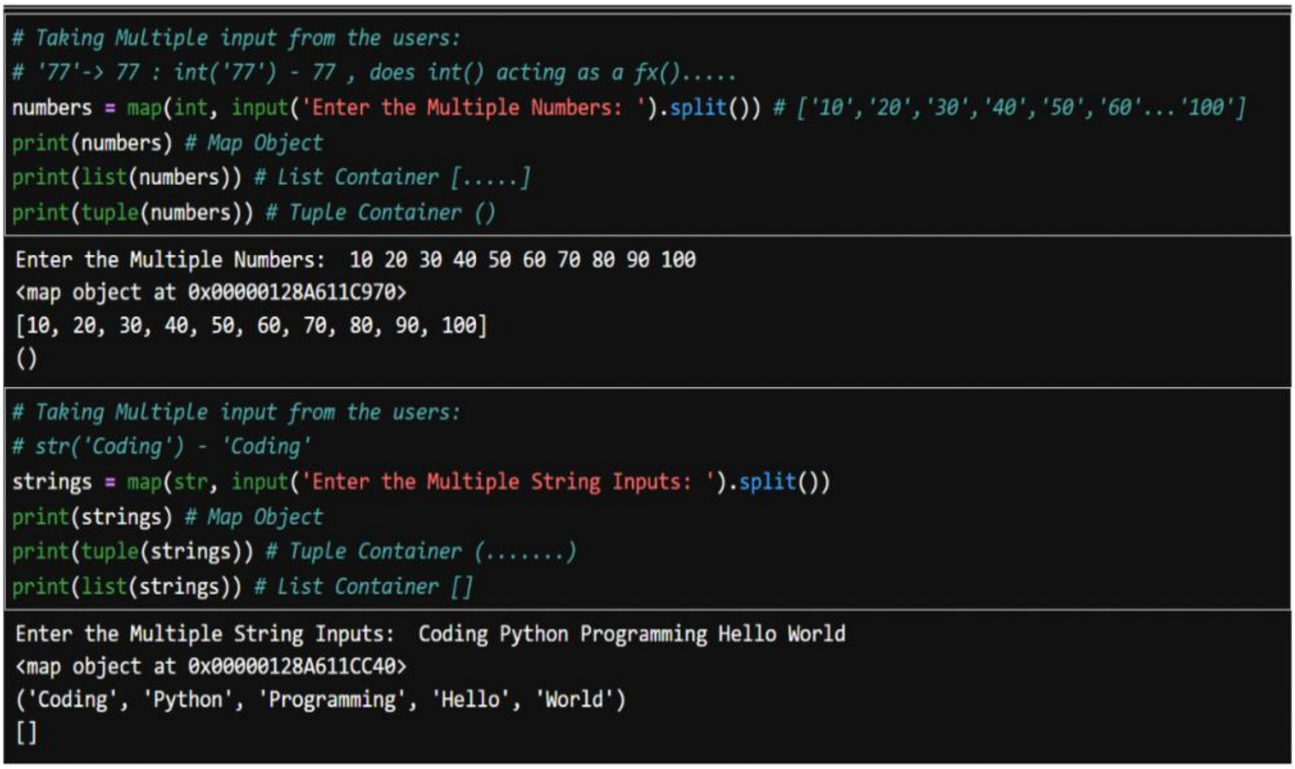
```
country_tuple = ('India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France',
                 'Croatia', 'Australia', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand')
# for i in range(1,6) # range(start,stop,step) # [1,2,3,4,5]
for i in range(len(country_tuple)): # [0,1,2,3,.......len-1]
    print(country_tuple[i] , end = '  ')
```

```
India  Russia  America  china  Canada  japan  Vietnam  Norway  France  Croatia  Australia  Finland  Pakistan
spain  Singapore  New-Zealand
```

```
country_tuple = ('India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France',
                 'Croatia', 'Australia', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand')
# for i in range(1,6) # range(start,stop,step) # [1,2,3,4,5]
country_list = []
for i in range(len(country_tuple)): # [0,1,2,3,.......len-1]
    country_list.append(country_tuple[i])

print(country_list)
```

```
['India', 'Russia', 'America', 'china', 'Canada', 'japan', 'Vietnam', 'Norway', 'France', 'Croatia', 'Australi
a', 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand']
```

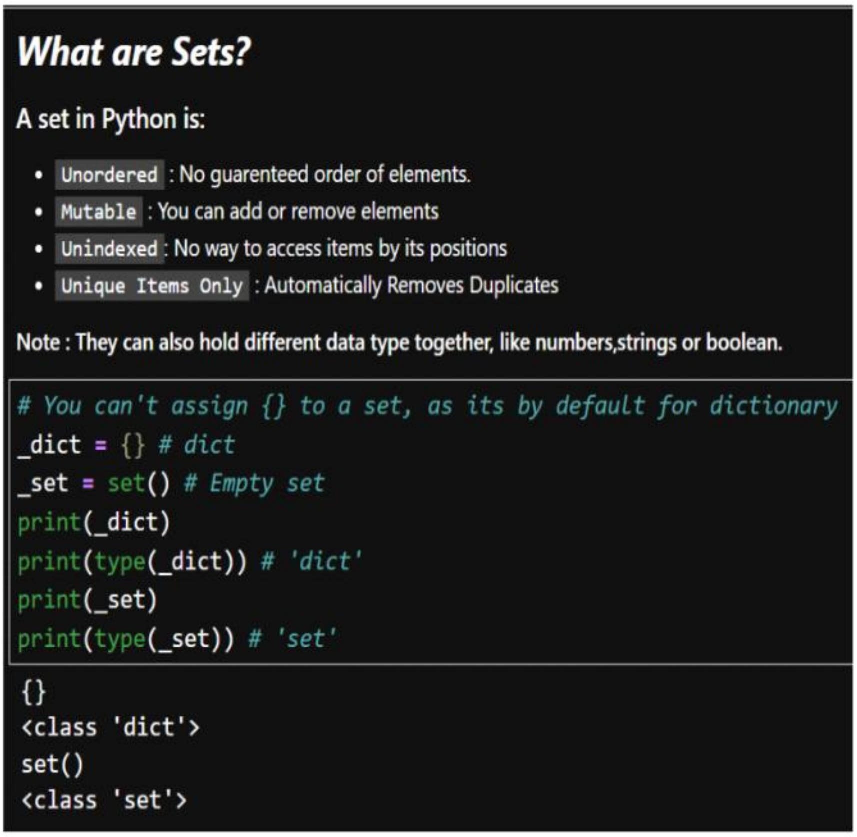


```
# Taking Multiple input from the users:
# '77'-> 77 : int('77') - 77 , does int() acting as a fx().....
numbers = map(int, input('Enter the Multiple Numbers: ').split()) # ['10','20','30','40','50','60'...'100']
print(numbers) # Map Object
print(list(numbers)) # List Container [.....]
print(tuple(numbers)) # Tuple Container ()
```

```
Enter the Multiple Numbers:  10 20 30 40 50 60 70 80 90 100
<map object at 0x00000128A611C970>
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
()
```

```
# Taking Multiple input from the users:
# str('Coding') - 'Coding'
strings = map(str, input('Enter the Multiple String Inputs: ').split())
print(strings) # Map Object
print(tuple(strings)) # Tuple Container (.......)
print(list(strings)) # List Container []
```

```
Enter the Multiple String Inputs:  Coding Python Programming Hello World
<map object at 0x00000128A611CC40>
('Coding', 'Python', 'Programming', 'Hello', 'World')
[]
```



What are Sets?

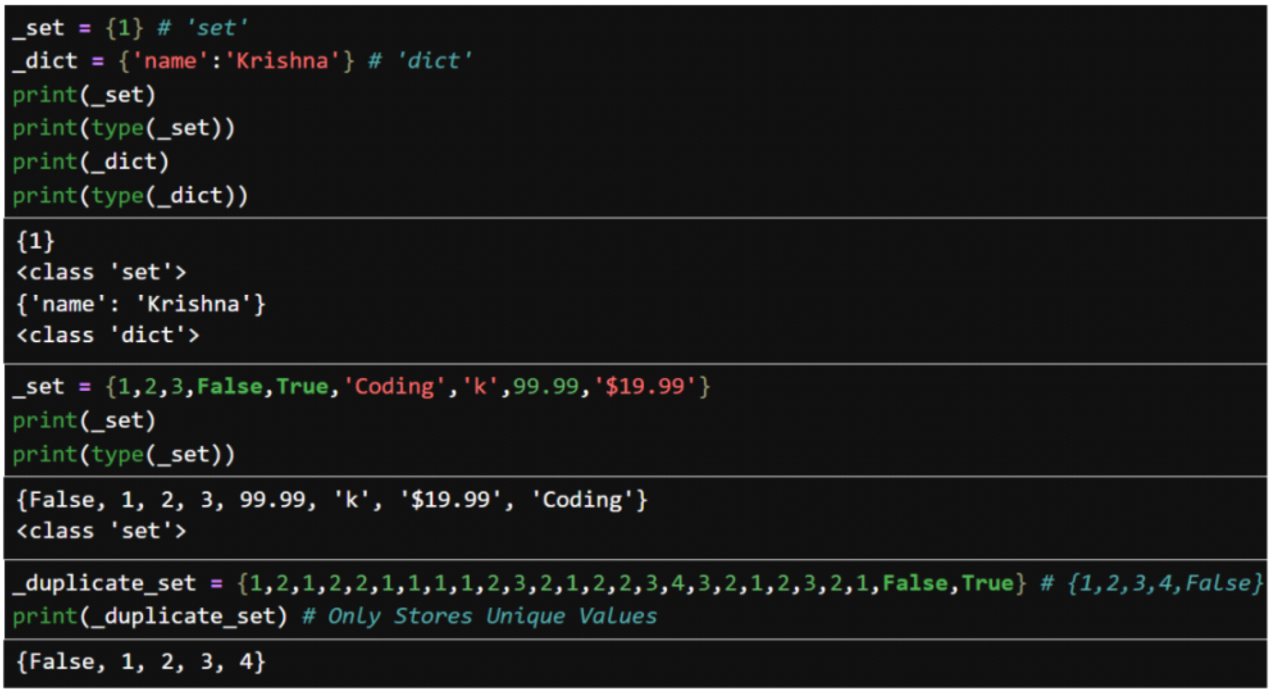
A set in Python is:

- `Unordered` : No guarenteed order of elements.
- `Mutable` : You can add or remove elements
- `Unindexed` : No way to access items by its positions
- `Unique Items Only` : Automatically Removes Duplicates

Note : They can also hold different data type together, like numbers,strings or boolean.

```
# You can't assign {} to a set, as its by default for dictionary
_dict = {} # dict
_set = set() # Empty set
print(_dict)
print(type(_dict)) # 'dict'
print(_set)
print(type(_set)) # 'set'
```

```
{}
<class 'dict'>
set()
<class 'set'>
```



```
_set = {1} # 'set'
_dict = {'name':'Krishna'} # 'dict'
print(_set)
print(type(_set))
print(_dict)
print(type(_dict))
```

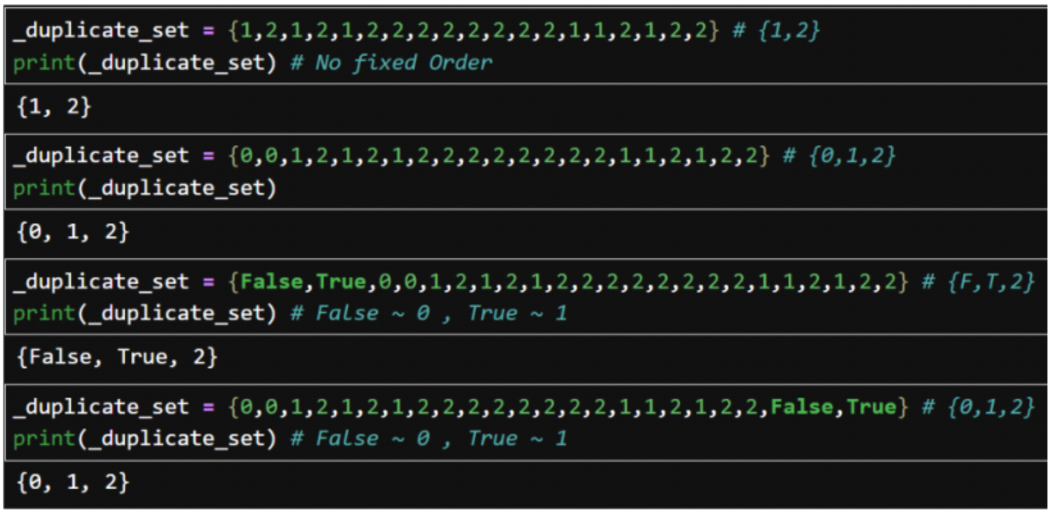
```
{1}
<class 'set'>
{'name': 'Krishna'}
<class 'dict'>
```

```
_set = {1,2,3,False,True,'Coding','k',99.99,'$19.99'}
print(_set)
print(type(_set))
```

```
{False, 1, 2, 3, 99.99, 'k', '$19.99', 'Coding'}
<class 'set'>
```

```
_duplicate_set = {1,2,1,2,2,1,1,1,1,2,3,2,1,2,2,3,4,3,2,1,2,3,2,1,False,True} # {1,2,3,4,False}
print(_duplicate_set) # Only Stores Unique Values
```

```
{False, 1, 2, 3, 4}
```



```
_duplicate_set = {1,2,1,2,1,2,2,2,2,2,2,2,2,1,1,2,1,2,2} # {1,2}
print(_duplicate_set) # No fixed Order
```

```
{1, 2}
```

```
_duplicate_set = {0,0,1,2,1,2,1,2,2,2,2,2,2,2,2,1,1,2,1,2,2} # {0,1,2}
print(_duplicate_set)
```

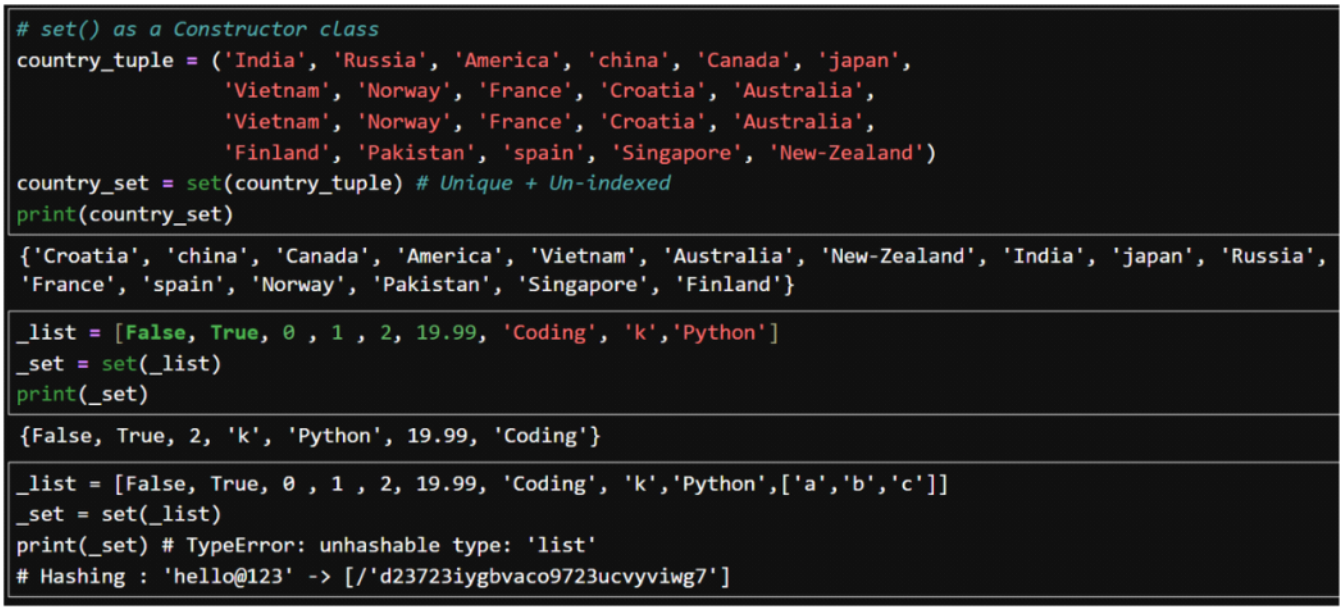
```
{0, 1, 2}
```

```
_duplicate_set = {False,True,0,0,1,2,1,2,1,2,2,2,2,2,2,2,2,1,1,2,1,2,2} # {F,T,2}
print(_duplicate_set) # False ~ 0 , True ~ 1
```

```
{False, True, 2}
```

```
_duplicate_set = {0,0,1,2,1,2,1,2,2,2,2,2,2,2,2,1,1,2,1,2,2,False,True} # {0,1,2}
print(_duplicate_set) # False ~ 0 , True ~ 1
```

```
{0, 1, 2}
```



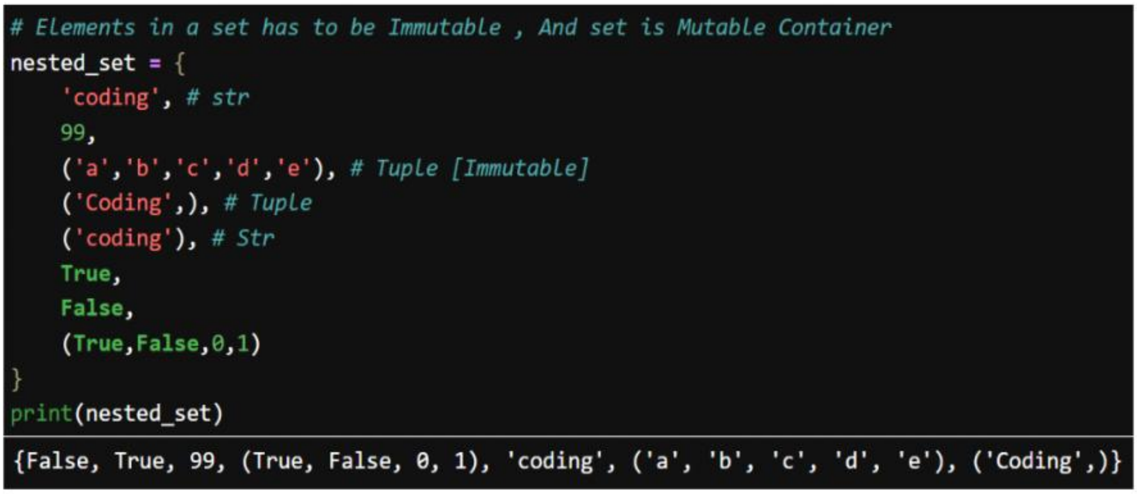
```
# set() as a Constructor class
country_tuple = ('India', 'Russia', 'America', 'china', 'Canada', 'japan',
                 'Vietnam', 'Norway', 'France', 'Croatia', 'Australia',
                 'Vietnam', 'Norway', 'France', 'Croatia', 'Australia',
                 'Finland', 'Pakistan', 'spain', 'Singapore', 'New-Zealand')
country_set = set(country_tuple) # Unique + Un-indexed
print(country_set)
```

```
{'Croatia', 'china', 'Canada', 'America', 'Vietnam', 'Australia', 'New-Zealand', 'India', 'japan', 'Russia',
'France', 'spain', 'Norway', 'Pakistan', 'Singapore', 'Finland'}
```

```
_list = [False, True, 0 , 1 , 2, 19.99, 'Coding', 'k','Python']
_set = set(_list)
print(_set)
```

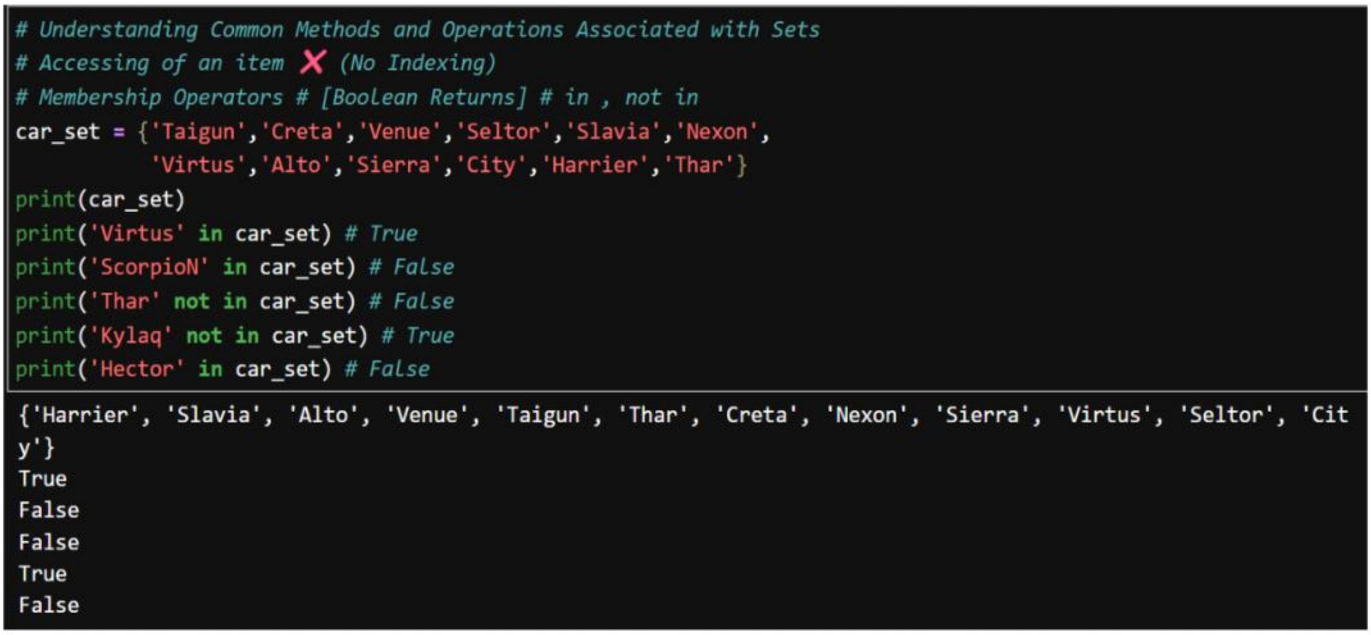
```
{False, True, 2, 'k', 'Python', 19.99, 'Coding'}
```

```
_list = [False, True, 0 , 1 , 2, 19.99, 'Coding', 'k','Python',['a','b','c']]
_set = set(_list)
print(_set) # TypeError: unhashable type: 'list'
# Hashing : 'hello@123' -> [/'d23723iygbvaco9723ucvyviwg7']
```



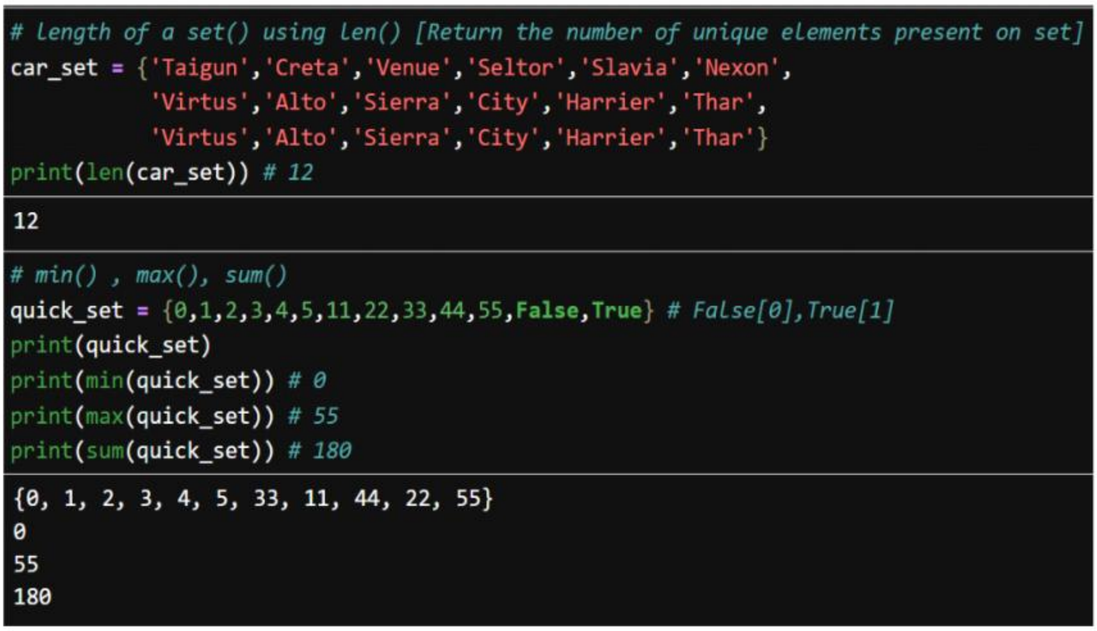
```python
# Elements in a set has to be Immutable , And set is Mutable Container
nested_set = {
    'coding', # str
    99,
    ('a','b','c','d','e'), # Tuple [Immutable]
    ('Coding',), # Tuple
    ('coding'), # Str
    True,
    False,
    (True,False,0,1)
}
print(nested_set)
```

```
{False, True, 99, (True, False, 0, 1), 'coding', ('a', 'b', 'c', 'd', 'e'), ('Coding',)}
```



```python
# Understanding Common Methods and Operations Associated with Sets
# Accessing of an item ❌ (No Indexing)
# Membership Operators # [Boolean Returns] # in , not in
car_set = {'Taigun','Creta','Venue','Seltor','Slavia','Nexon',
           'Virtus','Alto','Sierra','City','Harrier','Thar'}
print(car_set)
print('Virtus' in car_set) # True
print('ScorpioN' in car_set) # False
print('Thar' not in car_set) # False
print('Kylaq' not in car_set) # True
print('Hector' in car_set) # False
```

```
{'Harrier', 'Slavia', 'Alto', 'Venue', 'Taigun', 'Thar', 'Creta', 'Nexon', 'Sierra', 'Virtus', 'Seltor', 'Cit
y'}
True
False
False
True
False
```

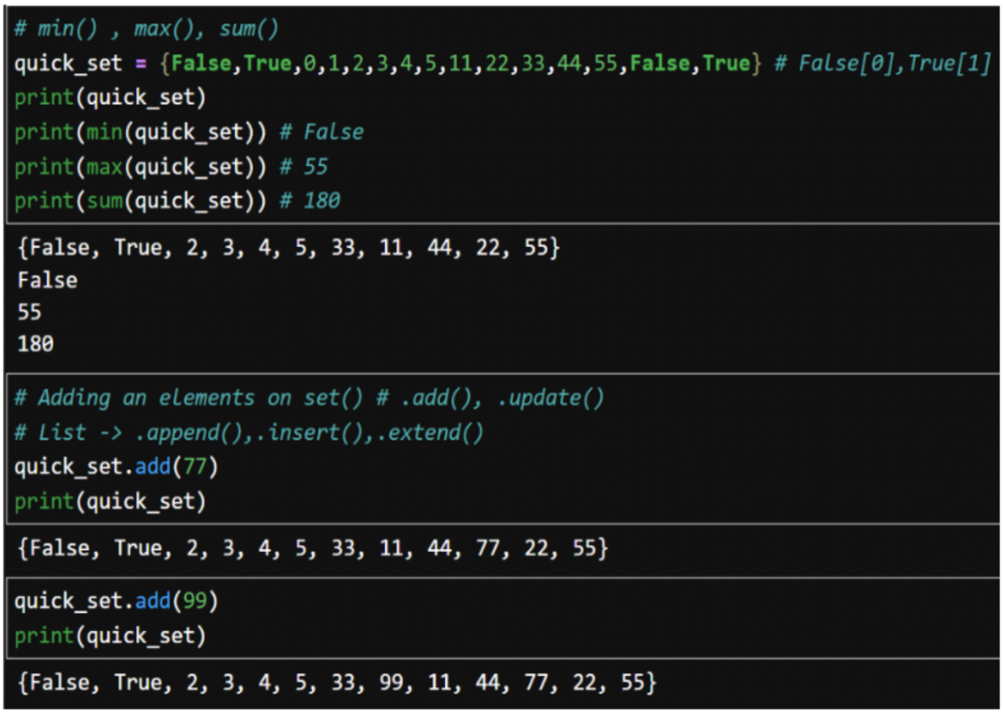


```python
# Length of a set() using len() [Return the number of unique elements present on set]
car_set = {'Taigun','Creta','Venue','Seltor','Slavia','Nexon',
           'Virtus','Alto','Sierra','City','Harrier','Thar',
           'Virtus','Alto','Sierra','City','Harrier','Thar'}
print(len(car_set)) # 12
```

```
12
```

```python
# min() , max(), sum()
quick_set = {0,1,2,3,4,5,11,22,33,44,55,False,True} # False[0],True[1]
print(quick_set)
print(min(quick_set)) # 0
print(max(quick_set)) # 55
print(sum(quick_set)) # 180
```

```
{0, 1, 2, 3, 4, 5, 33, 11, 44, 22, 55}
0
55
180
```



```
# min() , max(), sum()
quick_set = {False,True,0,1,2,3,4,5,11,22,33,44,55,False,True} # False[0],True[1]
print(quick_set)
print(min(quick_set)) # False
print(max(quick_set)) # 55
print(sum(quick_set)) # 180
```

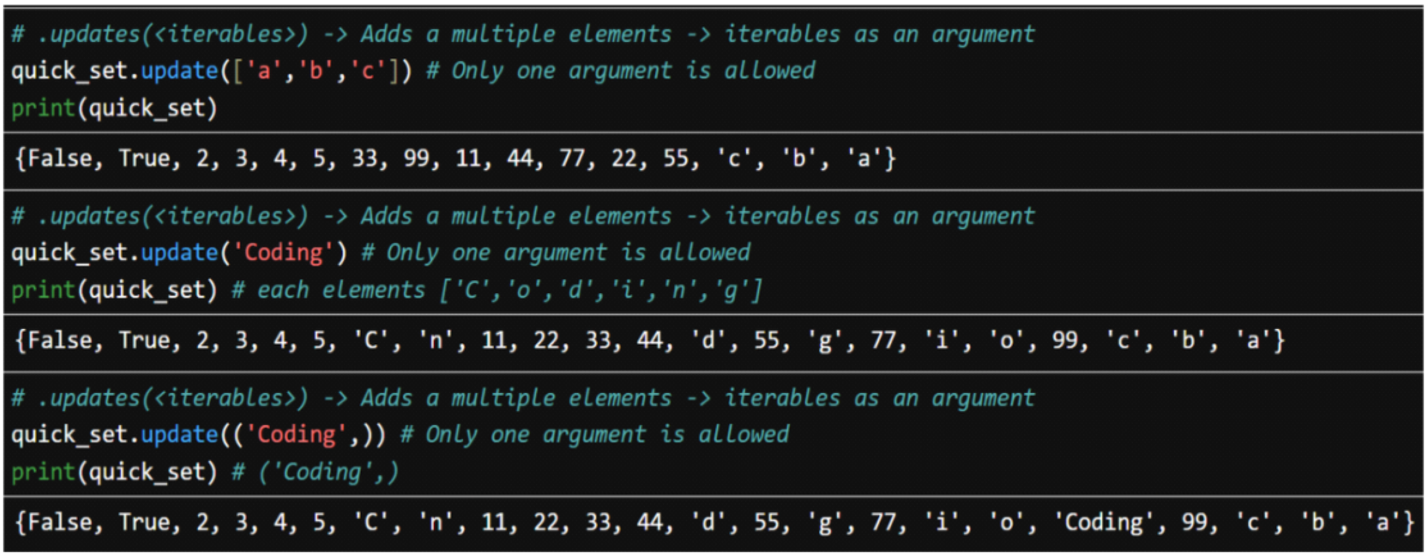
```
{False, True, 2, 3, 4, 5, 33, 11, 44, 22, 55}
False
55
180
```

```
# Adding an elements on set() # .add(), .update()
# List -> .append(),.insert(),.extend()
quick_set.add(77)
print(quick_set)
```

```
{False, True, 2, 3, 4, 5, 33, 11, 44, 77, 22, 55}
```

```
quick_set.add(99)
print(quick_set)
```

```
{False, True, 2, 3, 4, 5, 33, 99, 11, 44, 77, 22, 55}
```



```
# .updates(<iterables>) -> Adds a multiple elements -> iterables as an argument
quick_set.update(['a','b','c']) # Only one argument is allowed
print(quick_set)
```

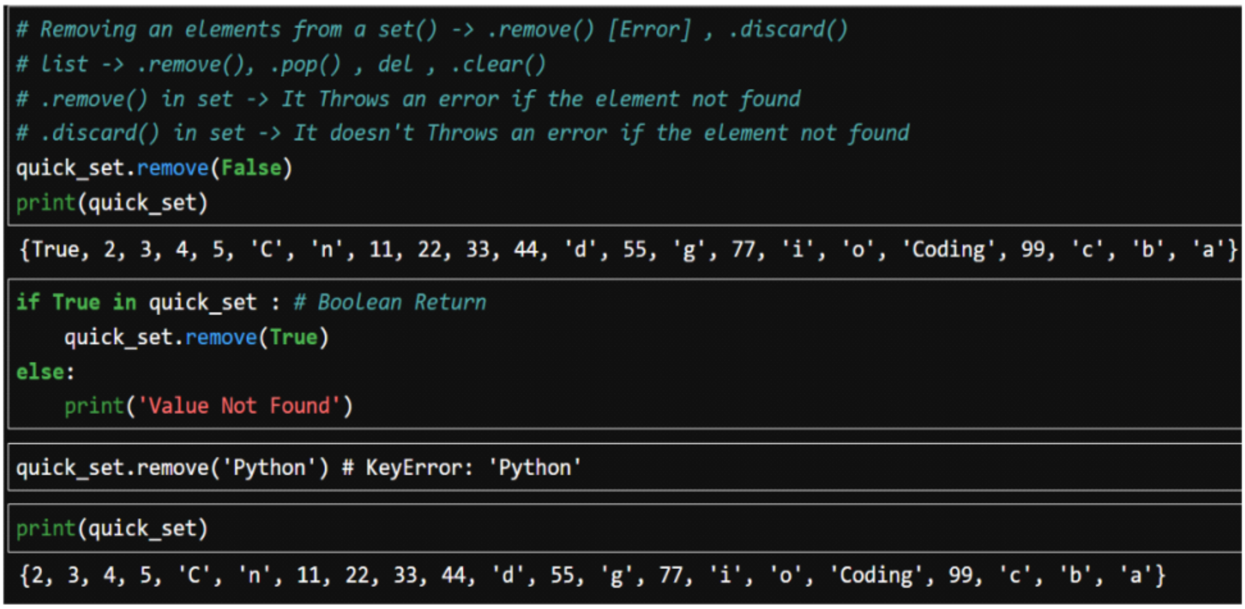
```
{False, True, 2, 3, 4, 5, 33, 99, 11, 44, 77, 22, 55, 'c', 'b', 'a'}
```

```
# .updates(<iterables>) -> Adds a multiple elements -> iterables as an argument
quick_set.update('Coding') # Only one argument is allowed
print(quick_set) # each elements ['C','o','d','i','n','g']
```

```
{False, True, 2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99, 'c', 'b', 'a'}
```

```
# .updates(<iterables>) -> Adds a multiple elements -> iterables as an argument
quick_set.update(('Coding',)) # Only one argument is allowed
print(quick_set) # ('Coding',)
```

```
{False, True, 2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 'Coding', 99, 'c', 'b', 'a'}
```



```
# Removing an elements from a set() -> .remove() [Error] , .discard()
# list -> .remove(), .pop() , del , .clear()
# .remove() in set -> It Throws an error if the element not found
# .discard() in set -> It doesn't Throws an error if the element not found
quick_set.remove(False)
print(quick_set)
```

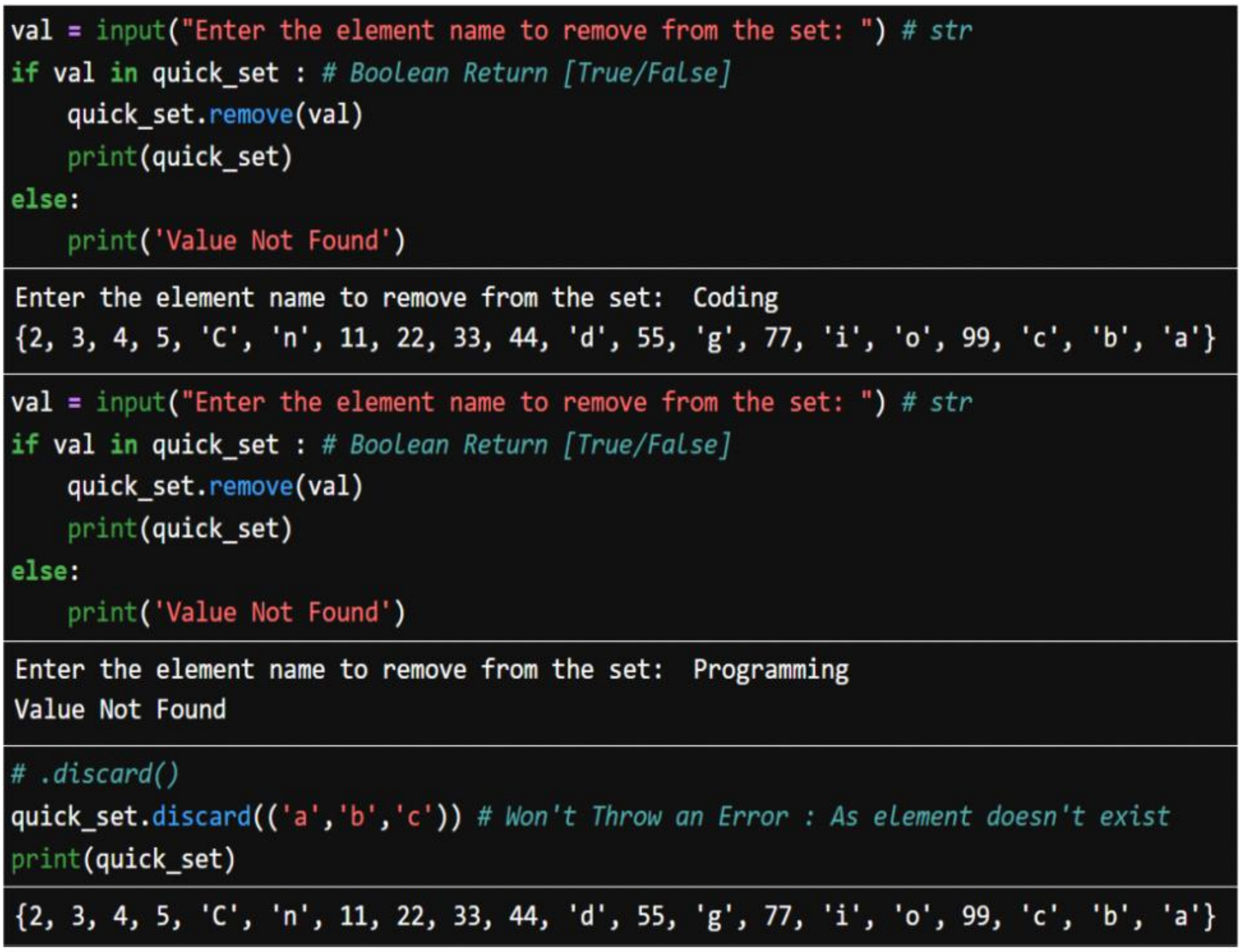
```
{True, 2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 'Coding', 99, 'c', 'b', 'a'}
```

```
if True in quick_set : # Boolean Return
    quick_set.remove(True)
else:
    print('Value Not Found')
```

```
quick_set.remove('Python') # KeyError: 'Python'
```

```
print(quick_set)
```

```
{2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 'Coding', 99, 'c', 'b', 'a'}
```



```
val = input("Enter the element name to remove from the set: ") # str
if val in quick_set : # Boolean Return [True/False]
    quick_set.remove(val)
    print(quick_set)
else:
    print('Value Not Found')
```

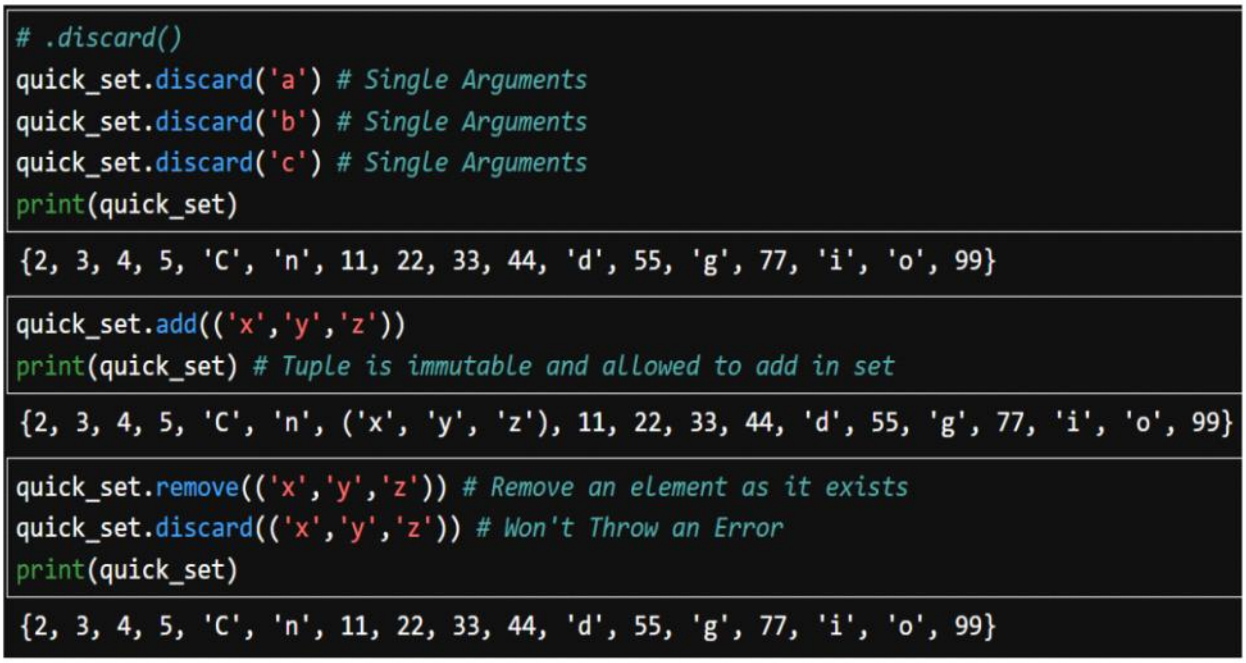
```
Enter the element name to remove from the set:  Coding
{2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99, 'c', 'b', 'a'}
```

```
val = input("Enter the element name to remove from the set: ") # str
if val in quick_set : # Boolean Return [True/False]
    quick_set.remove(val)
    print(quick_set)
else:
    print('Value Not Found')
```

```
Enter the element name to remove from the set:  Programming
Value Not Found
```

```
# .discard()
quick_set.discard(('a','b','c')) # Won't Throw an Error : As element doesn't exist
print(quick_set)
```

```
{2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99, 'c', 'b', 'a'}
```



```
# .discard()
quick_set.discard('a') # Single Arguments
quick_set.discard('b') # Single Arguments
quick_set.discard('c') # Single Arguments
print(quick_set)
```

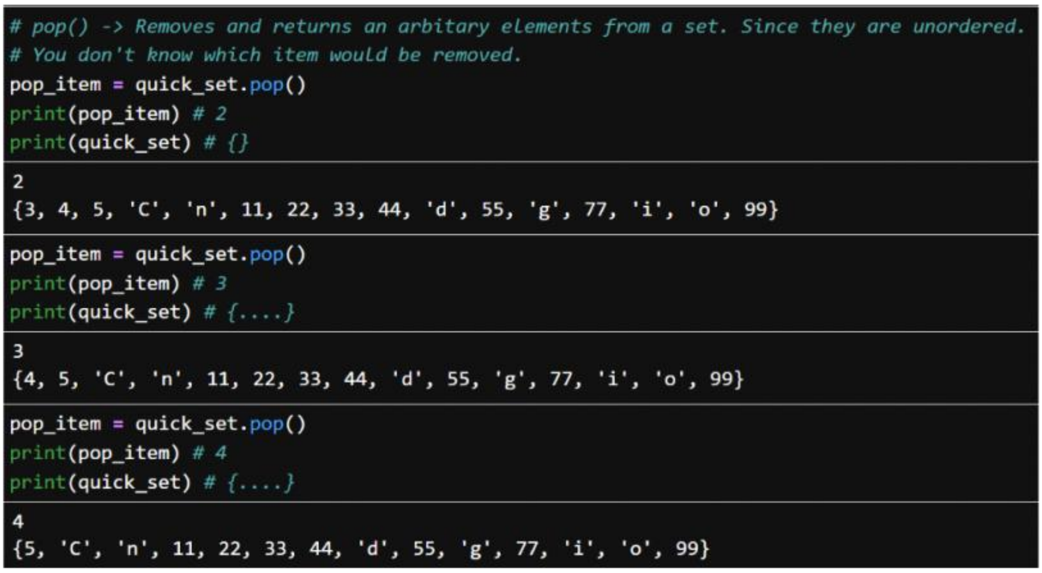
```
{2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```

```
quick_set.add(('x','y','z'))
print(quick_set) # Tuple is immutable and allowed to add in set
```

```
{2, 3, 4, 5, 'C', 'n', ('x', 'y', 'z'), 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```

```
quick_set.remove(('x','y','z')) # Remove an element as it exists
quick_set.discard(('x','y','z')) # Won't Throw an Error
print(quick_set)
```

```
{2, 3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```



```
# pop() -> Removes and returns an arbitary elements from a set. Since they are unordered.
# You don't know which item would be removed.
pop_item = quick_set.pop()
print(pop_item) # 2
print(quick_set) # {}
```

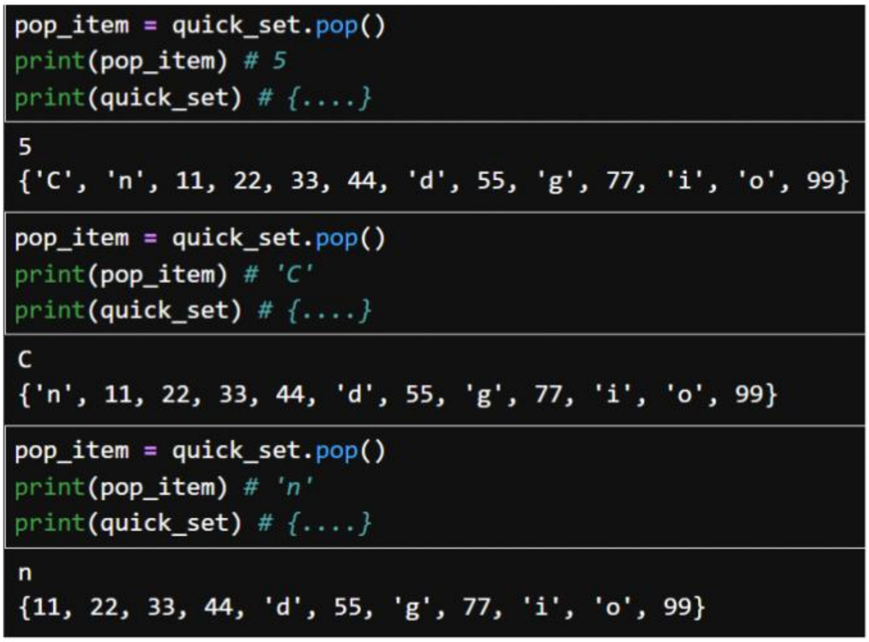
```
2
{3, 4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```

```
pop_item = quick_set.pop()
print(pop_item) # 3
print(quick_set) # {....}
```

```
3
{4, 5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```

```
pop_item = quick_set.pop()
print(pop_item) # 4
print(quick_set) # {....}
```

```
4
{5, 'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```



```
pop_item = quick_set.pop()
print(pop_item) # 5
print(quick_set) # {....}
```

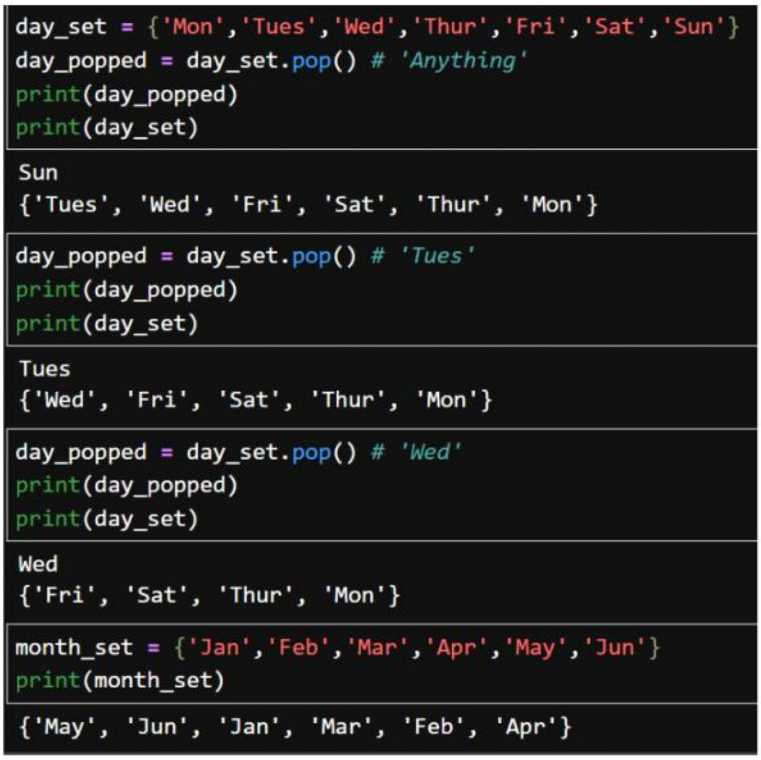
```
5
{'C', 'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```

```
pop_item = quick_set.pop()
print(pop_item) # 'C'
print(quick_set) # {....}
```

```
C
{'n', 11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```

```
pop_item = quick_set.pop()
print(pop_item) # 'n'
print(quick_set) # {....}
```

```
n
{11, 22, 33, 44, 'd', 55, 'g', 77, 'i', 'o', 99}
```



```
day_set = {'Mon','Tues','Wed','Thur','Fri','Sat','Sun'}
day_popped = day_set.pop() # 'Anything'
print(day_popped)
print(day_set)
```

```
Sun
{'Tues', 'Wed', 'Fri', 'Sat', 'Thur', 'Mon'}
```

```
day_popped = day_set.pop() # 'Tues'
print(day_popped)
print(day_set)
```

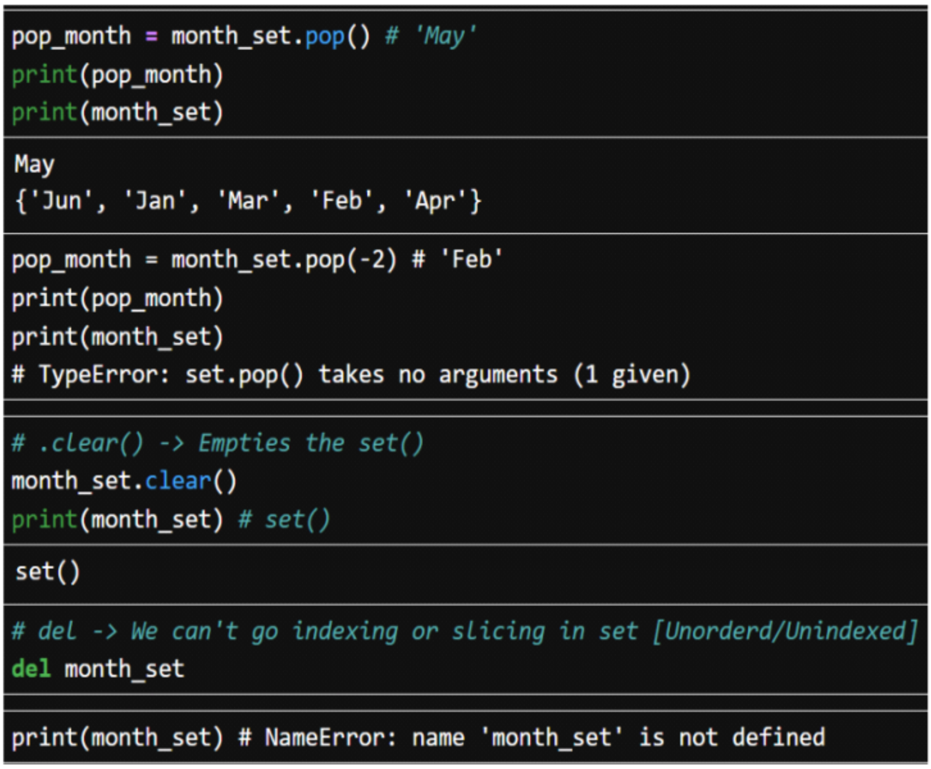
```
Tues
{'Wed', 'Fri', 'Sat', 'Thur', 'Mon'}
```

```
day_popped = day_set.pop() # 'Wed'
print(day_popped)
print(day_set)
```

```
Wed
{'Fri', 'Sat', 'Thur', 'Mon'}
```

```
month_set = {'Jan','Feb','Mar','Apr','May','Jun'}
print(month_set)
```

```
{'May', 'Jun', 'Jan', 'Mar', 'Feb', 'Apr'}
```



```
pop_month = month_set.pop() # 'May'
print(pop_month)
print(month_set)
```

```
May
{'Jun', 'Jan', 'Mar', 'Feb', 'Apr'}
```

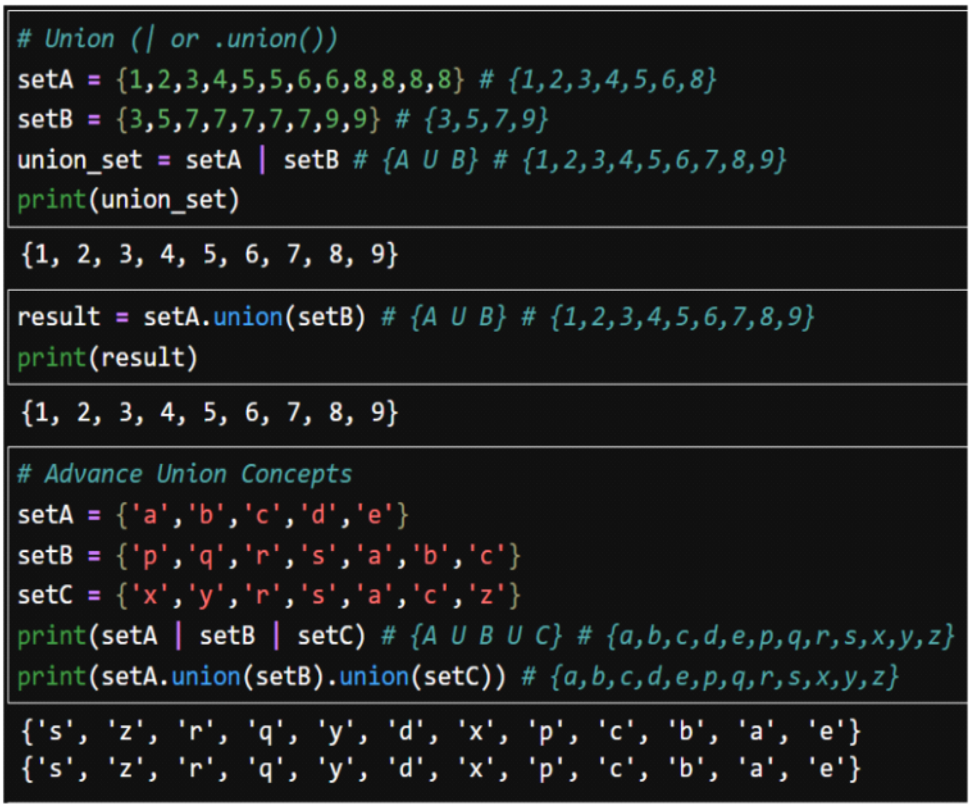
```
pop_month = month_set.pop(-2) # 'Feb'
print(pop_month)
print(month_set)
# TypeError: set.pop() takes no arguments (1 given)
```

```
# .clear() -> Empties the set()
month_set.clear()
print(month_set) # set()
```

```
set()
```

```
# del -> We can't go indexing or slicing in set [Unorderd/Unindexed]
del month_set
```

```
print(month_set) # NameError: name 'month_set' is not defined
```



```
# Union (| or .union())
setA = {1,2,3,4,5,5,6,6,8,8,8,8} # {1,2,3,4,5,6,8}
setB = {3,5,7,7,7,7,7,9,9} # {3,5,7,9}
union_set = setA | setB # {A U B} # {1,2,3,4,5,6,7,8,9}
print(union_set)
```

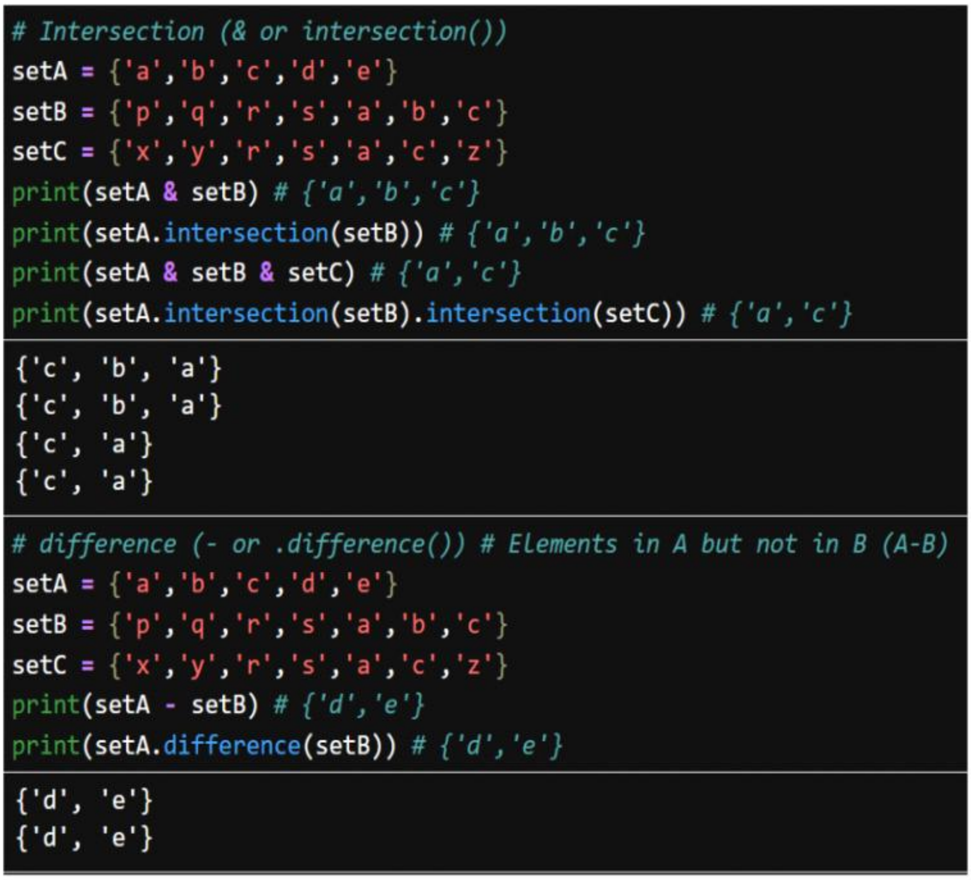
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
result = setA.union(setB) # {A U B} # {1,2,3,4,5,6,7,8,9}
print(result)
```

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

```
# Advance Union Concepts
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setA | setB | setC) # {A U B U C} # {a,b,c,d,e,p,q,r,s,x,y,z}
print(setA.union(setB).union(setC)) # {a,b,c,d,e,p,q,r,s,x,y,z}
```

```
{'s', 'z', 'r', 'q', 'y', 'd', 'x', 'p', 'c', 'b', 'a', 'e'}
{'s', 'z', 'r', 'q', 'y', 'd', 'x', 'p', 'c', 'b', 'a', 'e'}
```

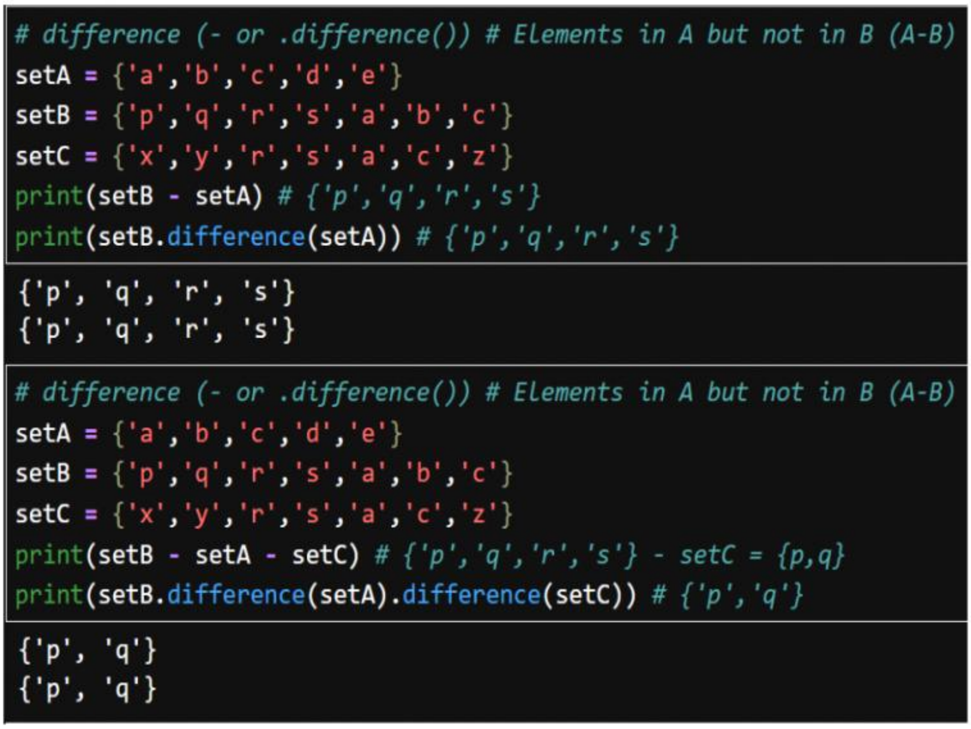


```python
# Intersection (& or intersection())
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setA & setB) # {'a','b','c'}
print(setA.intersection(setB)) # {'a','b','c'}
print(setA & setB & setC) # {'a','c'}
print(setA.intersection(setB).intersection(setC)) # {'a','c'}
```

```
{'c', 'b', 'a'}
{'c', 'b', 'a'}
{'c', 'a'}
{'c', 'a'}
```

```python
# difference (- or .difference()) # Elements in A but not in B (A-B)
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setA - setB) # {'d','e'}
print(setA.difference(setB)) # {'d','e'}
```

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

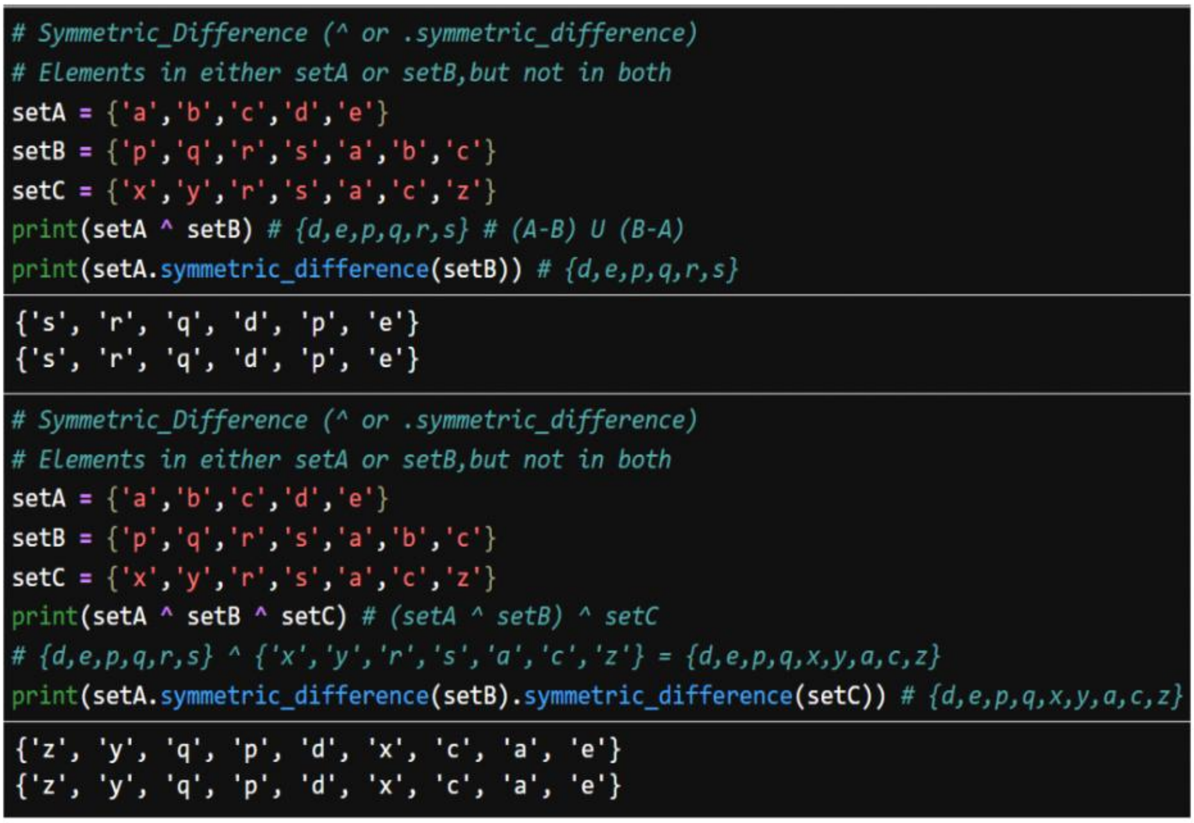


```python
# difference (- or .difference()) # Elements in A but not in B (A-B)
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setB - setA) # {'p','q','r','s'}
print(setB.difference(setA)) # {'p','q','r','s'}
```

```
{'p', 'q', 'r', 's'}
{'p', 'q', 'r', 's'}
```

```python
# difference (- or .difference()) # Elements in A but not in B (A-B)
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setB - setA - setC) # {'p','q','r','s'} - setC = {p,q}
print(setB.difference(setA).difference(setC)) # {'p','q'}
```

```
{'p', 'q'}
{'p', 'q'}
```



```
# Symmetric_Difference (^ or .symmetric_difference)
# Elements in either setA or setB,but not in both
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setA ^ setB) # {d,e,p,q,r,s} # (A-B) U (B-A)
print(setA.symmetric_difference(setB)) # {d,e,p,q,r,s}
```

```
{'s', 'r', 'q', 'd', 'p', 'e'}
{'s', 'r', 'q', 'd', 'p', 'e'}
```

```
# Symmetric_Difference (^ or .symmetric_difference)
# Elements in either setA or setB,but not in both
setA = {'a','b','c','d','e'}
setB = {'p','q','r','s','a','b','c'}
setC = {'x','y','r','s','a','c','z'}
print(setA ^ setB ^ setC) # (setA ^ setB) ^ setC
# {d,e,p,q,r,s} ^ {'x','y','r','s','a','c','z'} = {d,e,p,q,x,y,a,c,z}
print(setA.symmetric_difference(setB).symmetric_difference(setC)) # {d,e,p,q,x,y,a,c,z}
```

```
{'z', 'y', 'q', 'p', 'd', 'x', 'c', 'a', 'e'}
{'z', 'y', 'q', 'p', 'd', 'x', 'c', 'a', 'e'}
```