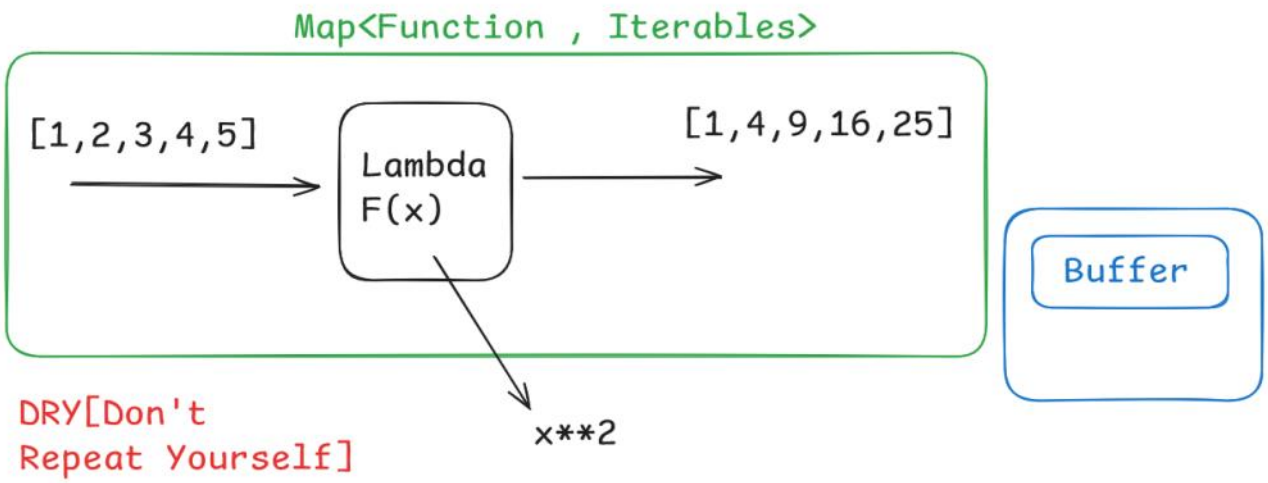
Data Structures-II

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.



DRY[Don't
Repeat Yourself]

X^2+2x+3 => X(1),X(2)

```
# Square Each Elements
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(tuple(squared)) # (1,4,9,16,25,36,49)
print(list(squared)) # []
```

```
<map object at 0x000001610BBD04F0>
(1, 4, 9, 16, 25, 36, 49)
[]
```

```python
# Immutability of Tuples...
# We can't change or add any elements directly.
# Tuples don't have: - append, extend, remove, insert
# But we can do it indirectly by modifying it into list
country_tuple = ('America','India', 'Russia', 'China', 'Canada', 'Japan',
                 'Vietnam', 'Sri-Lanka','India', 'france', 'singapore', 'australia',
                 'Spain', 'New Zealand', 'Finland')
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'>
['America', 'India', 'Russia', 'China', 'Canada', 'Japan', 'Vietnam', 'Sri-Lanka', 'India', 'france', 'singapo
re', 'australia', 'Spain', 'New Zealand', 'Finland', 'Germany']
```

```python
country_tuple = tuple(conv_tuple_to_list)
print(country_tuple)
print(type(country_tuple))
```

```
('America', 'India', 'Russia', 'China', 'Canada', 'Japan', 'Vietnam', 'Sri-Lanka', 'India', 'france', 'singapo
re', 'australia', 'Spain', 'New Zealand', 'Finland', 'Germany')
<class 'tuple'>
```

```python
# Concatenating a tuples
weekday_tuple = ('Mon','Tues','Wed','Thur','Fri')
weekend_tuple = ('Sat','Sun')
week_tuple = weekday_tuple + weekend_tuple
print(week_tuple)
```

```
('Mon', 'Tues', 'Wed', 'Thur', '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
# week_tuple[0] # ['Mon', 'Tues']
week_tuple[0].extend(['Wed','Thurs','Fri'])
print(week_tuple)
```

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

```python
weekday_tuple = ('Mon','Tues','Wed','Thur','Fri')
weekend_tuple = ('Sat','Sun')
week_tuple = weekday_tuple[:3] + weekend_tuple
print(week_tuple)
```

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

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

```
# Deleting an element in Tuple VS Deleting a Tuple
# You can't delete individual elements in a tuple with 'del'
# But we can delete the entire tuple object
weekday_tuple = ('Mon','Tues','Wed','Thur','Fri')
print(weekday_tuple)
print(type(weekday_tuple)) # 'tuple'
```

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

```
del weekday_tuple[3] # 'Thurs' # TypeError: 'tuple' object doesn't support item deletion
```

```
del weekday_tuple # This will delete tuple from the memory
# print(weekday_tuple) # NameError: name 'weekday_tuple' is not defined
```

```
# Unpacking a Tuple
a,b,c = (1,2,3)
print(a) # 1
print(b) # 2
print(c) # 3
```

```
1
2
3
```

```
a,*b,c = (1,2,3,4,5,6,7,8,9)
print(a) # 1
print(b) # [2,3,4,5,6,7,8] ['List']
print(c) # 9
```

```
1
[2, 3, 4, 5, 6, 7, 8]
9
```

```
a,*b,*c = (1,2,3,4,5,6,7,8,9) # SyntaxError: multiple starred expressions in assignment
print(a)
print(b)
print(c)
```

```
*a,b,c = (1,2,3,4,5,6,7,8,9)
print(a) # [1,2,3,4,5,6,7]
print(tuple(a)) # (1,2,3,4,5,6,7)
print(b) # 8
print(c) # 9
```

```
[1, 2, 3, 4, 5, 6, 7]
(1, 2, 3, 4, 5, 6, 7)
8
9
```

```
a,b,c = (1,2,3,4,5,6,7,8,9) # ValueError: too many values to unpack (expected 3)
print(a)
print(b)
print(c)
```

```
# Repeat ['*']
_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 - '+'. Tuple Contains Duplicates Elements and also store mixed data types
_tuple1 = (1,2,3,4,5) # Homogeneous Data Type
_tuple2 = (False,True,19.99,'Coding','a','b','c','Python@123') # Heterogenous Data Type
new_tuple = _tuple1 + _tuple2
print(new_tuple)
```

```
(1, 2, 3, 4, 5, False, True, 19.99, 'Coding', 'a', 'b', 'c', 'Python@123')
```

```
country_tuple = ('America','India', 'Russia', 'China', 'Canada', 'Japan',
                 'America','India', 'Russia', 'China', 'Canada', 'Japan',
                'Vietnam', 'Sri-Lanka','India', 'france', 'singapore', 'australia',
                'Spain', 'New Zealand', 'Finland')
country_tuple.count('India') # 3
```

```
3
```

```
# .index() - returns the positions
country_tuple = ('America','India', 'Russia', 'China', 'Canada', 'Japan',
                 'America','India', 'Russia', 'China', 'Canada', 'Japan',
                'Vietnam', 'Sri-Lanka','India', 'france', 'singapore', 'australia',
                'Spain', 'New Zealand', 'Finland')
country_tuple.index('Japan') # First Occurrence # 5
```

```
5
```

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

Using Map() in Python:

What is Map()?

`Syntax: map(function , Iterables)`

- where function uses lambda function internally
- The map() function is a very handy tool in Python that lets you:
  - Apply a Function to each item in a list, tuple or any other iterables
  - Returns a map object

```python
# Square Each Elements
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(tuple(squared)) # (1,4,9,16,25,36,49)
print(list(squared)) # []
```

```
<map object at 0x000001610BBD04F0>
(1, 4, 9, 16, 25, 36, 49)
[]
```

```python
# Add 10 to the 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(list(add_10)) # [11,12,13,14,15,16,17]
print(tuple(add_10)) # ()
```

```
<map object at 0x000001610BBD2320>
[11, 12, 13, 14, 15, 16, 17]
()
```

```
INSERT INTO TABLE_NAME(Col1,Col2,Col3,Col4) # Parameters
VALUES(arg1,arg2,arg3,arg4) # Arguments
```

```python
# Taking a input from the user : <Multiple>
_bool = True
int_bool = int(_bool) # int(True) # 1
# # map(function , Iterables)
numbers = list(map(int,input('Enter the Multiple numbers: ').split()))
print(numbers) # list
```

```
Enter the Multiple numbers:  10 20 30 40 50 60 70 80 90 100
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
input('Enter the Multiple numbers: ').split()) -> ['10','20','30','40','50','60','70','80','90','100'] # str
# It is returning an Iterables
Map(Function , Iterables) # Function - int()
```

```python
# Taking an input from the users:
# map(function , Iterables)
string_tuple = tuple(map(str,input('Enter the Multiple Strings : ').split()))
print(string_tuple) # tuple
```

```
Enter the Multiple Strings :  Coding Ninja Python Programming
('Coding', 'Ninja', 'Python', 'Programming')
```

```python
# Taking an input from the users:
# map(function , Iterables)
string_tuple = tuple(map(str,input('Enter the Multiple Strings : ').split(',')))
print(string_tuple) # tuple
```

```
Enter the Multiple Strings :  Coding,Ninja,Python,Programming
('Coding', 'Ninja', 'Python', 'Programming')
```

What are Sets?

A Set in Python is:

- Unordered : No guarenteed order of elements
- Changeable(mutable) : You can add or remove elements
- Unindexed : No way to access items by its position
- Stores Unique Items Only : Automatically removes duplicates

They can hold different data types together, like numbers, strings or boolean.

```python
# List [] , Tuple () , set {} ❌ , Dictionary {} ✅
_dict = {} # Empty Dictionary
_set = set() # Set Constructor class is an empty set
print(_dict) # {}
print(type(_dict)) # 'dict'
print(_set) # set()
print(type(_set)) # 'set'
```

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

```python
_set = {1} # 'set'
print(_set)
print(type(_set))
```

```
{1}
<class 'set'>
```

```python
# No Duplicates , Unordered
_set = {1,2,3,'a','b','c',False,True,9.99,'Coding'}
print(_set)
print(type(_set))
```

```
{False, 1, 2, 3, 'c', 9.99, 'a', 'Coding', 'b'}
<class 'set'>
```

```python
_set = {1,2,3,'a','b','c',False,True,9.99,'Coding','python','programming'}
print(_set)
print(type(_set))
```

```
{False, 1, 2, 3, 'programming', 'c', 9.99, 'a', 'python', 'Coding', 'b'}
<class 'set'>
```

```python
_duplicate_sets = {1,2,3,2,1,2,1,2,1,2,2,1,11}
print(_duplicate_sets) # {1,2,3,11} # final answer will be shuffled
```

```
{3, 1, 2, 11}
```

```python
_duplicate_sets = {0,1,0,1,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,0,False,True}
print(_duplicate_sets) # Only Stores unique Elements # {1,0}
```

```
{0, 1}
```

```python
_duplicate_sets = {False,True,0,1,0,1,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,0,False,True}
print(_duplicate_sets) # Only Stores unique Elements # {False,True}
```

```
{False, True}
```

```python
# Calling set() constructor to do the typecasting
country_tuple = ('America','India', 'Russia', 'China', 'Canada', 'Japan',
                 'America','India', 'Russia', 'China', 'Canada', 'Japan',
                'Vietnam', 'Sri-Lanka','India', 'france', 'singapore', 'australia',
                'Spain', 'New Zealand', 'Finland')
_list = [False,True,19.99,'Coding','a','b','c',0,1,2,3,4,5]
print(set(country_tuple)) # {unique + Shuffle}
print(set(_list)) # {unique + Shuffle}
```

```
{'Finland', 'Sri-Lanka', 'france', 'Canada', 'Russia', 'Vietnam', 'America', 'Spain', 'singapore', 'New Zealan
d', 'China', 'India', 'Japan', 'australia'}
{False, True, 2, 3, 4, 5, 'c', 19.99, 'a', 'Coding', 'b'}
```

```python
# Nested_Set : Immutable elements are allowed in sets
nested_set = {
    'coding',
    99,
    ('a','b','c','d'),
    ('coding',),
    ('coding'),
    (True,False)
}
nested_set
```

```
{('a', 'b', 'c', 'd'), ('coding',), (True, False), 99, 'coding'}
```

Key Takeaways for Sets: ¶

1. Unordered: You can't access elements by index.
2. Unique: Removes Duplicates Automatically.
3. Mutable Container : Can add or remove items.
4. Immutable Elements : Elements must be hashable & immutable(like numbers, strings, tuples)

```python
# Understanding Common Methods and operations associated with sets..
# Accessing an item [❌ No Indexing]
# Membership Operators [Boolean Returns]
car_set = {'Taigun','Creta','Thar','Kylaq','Camry','Seirra','Sonet','Venue','Nexon','Defender','Verna'}
print('Slavia' in car_set) # False
print('Thar' in car_set) # True
print('Kylaq' not in car_set) # False
print('ScorpioN' not in car_set) # True
```

```
False
True
False
True
```

```python
# Length of a set -> len() returning count the number of elements.
car_set = {'Taigun','Creta','Thar','Kylaq','Camry','Seirra','Sonet','Venue','Nexon','Defender','Verna'}
print(len(car_set)) # 11
```

```
11
```

```python
# Length of a set -> len() returning count the number of elements.
car_set = {'Taigun','Creta','Thar','Kylaq','Camry','Seirra','Sonet','Venue','Nexon','Defender','Verna',
           'Taigun','Creta','Thar','Kylaq','Camry','Seirra','Sonet','Venue','Nexon','Defender','Verna'}
print(len(car_set)) # 11
```

```
11
```

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

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

```python
# Adding Items to a set() [Mutable Containers]
# .add() -> Adds a single elements
quick_set = {False,True,2,3,4,5,11,22,33,44,55,0,1}
quick_set.add(77) # Added anywhere
print(quick_set)
```

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

```python
print(min(quick_set)) # False
print(max(quick_set)) # 77
print(sum(quick_set)) # 257
```

```
False
77
257
```

```
quick_set.add(99) # Added anywhere
print(quick_set)
```

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

```
quick_set.add(99,11,22) # TypeError: set.add() takes exactly one argument (3 given)
print(quick_set)
```

```
#.update() -> Multiple Elements will be inserted [Iterable]
# .update(<iterables>) -> Adds a multiple elements -> Passing Iterables as an argument
quick_set.update(['a','b','c']) # 'Unique' and doesn't pre-exist.
print(quick_set)
```

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

```
quick_set.update(['a','b','c','d','e','Coding'])
print(quick_set)
```

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

```
quick_set.update('Coding') # ['C','o','d','i','n','g']
print(quick_set)
```

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

```
quick_set.update(('Python',)) # ('Python',)
print(quick_set)
```

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