



Django for Web Development & Artificial Intelligence

Lecture - 04

Instructor:

Md. Abu Noman Basar
Django Developer



Python Tuple

- ❑ A tuple is a collection that is ordered and **unchangeable**. Tuples are written with round brackets ().

Ex:

```
details = ('create', 'read', 'update', 'delete')  
print(type(details))
```

❑ Access item

Ex:

```
details = ('create', 'read', 'update', 'delete')  
print('Access item: ', details[2])
```

Python Tuple

❑ Update

- Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
- You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Ex:

```
details = ('create', 'read', 'update', 'delete')
cd = list(details)
cd[2] = 'Django'
details = tuple(cd)
print('Update item: ', details)
```

Python Tuple

❑ Add item

- Convert the tuple into a list, change the list, and convert the list back into a tuple.

Ex:

```
details = ('create', 'read', 'update', 'delete')
cd = list(details)
cd.append('Django')
details = tuple(cd)
print('Add item: ', details)
```

❑ Remove item

Ex:

```
details = ('create', 'read', 'update', 'delete')
cd = list(details)
cd.remove('delete')
details = tuple(cd)
print('Remove item: ', details)
```



Python Set

❑ A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements.

- Sets are written with curly brackets.

Ex:

```
duration = {2,5,7,9,55,1}
print(type(duration))
```

❑ **Not allow duplicate item**

Ex:

```
duration = {2,5,7,9,55,1,55,9}
print('Not allow duplicate value',duration)
```

❑ 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.

Ex:

```
duration = {2,5,7,9,55,1,55,9}
for x in duration:
    print('Access item: ',x)
```

❑ add() method

Ex:

```
duration = {2,5,7,9,55,1,55,9}
duration.add(99)
print('Add item', duration)
```

❑ discard() method

Ex:

```
duration = {2,5,7,9,55,1,55,9}
duration.discard(9)
print('Discard item', duration)
```

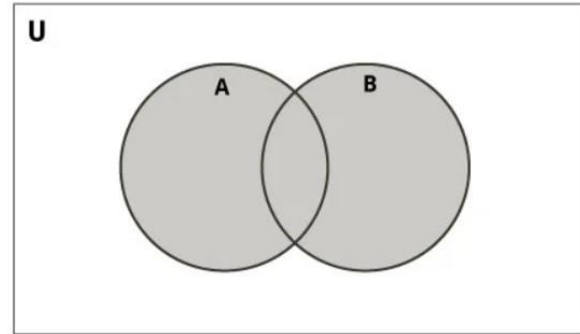
❑ del keyword

Ex:

```
duration = {2,5,7,9,55,1,55,9}
del duration
print('Delete Set', duration)
```

Python Set

□ Set Union



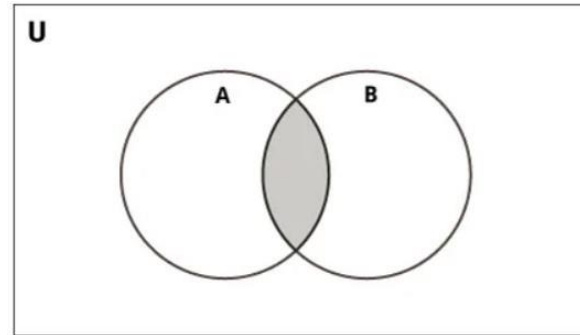
- Union of **A** and **B** is a set of all elements from both sets. Union performs using `|`

Ex:

```
duration = {2, 5, 7, 9, 55, 6}
success = {44, 55, 100, 6, 2}
print('Union method', duration | success)
```


Python Set

❑ Set Intersection



- Intersection of **A** and **B** is a set of all elements that are common in both sets. Union performs using **&**

Ex:

```
duration = {2, 5, 7, 9, 55, 6}
success = {44, 55, 100, 6, 2}
print('Intersection method', duration & success)
```



Python Dictionary

- ❑ **Dictionary** is a collection of key values, used to store data values like a map, which, unlike other data types holds only a single value as an element.
- ❑ Creating a dictionary is as simple as placing items inside curly braces `{}` separated by commas.
- ❑ An item has a **key** and a corresponding **value** that is expressed as a pair (key: value)

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
print(Dict)
```



Python Dictionary

❑ Accessing Items

❑ keys()

The **keys()** method will return a list of all the keys in the dictionary.

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
x = Dict.keys()  
print(x)
```

❑ values()

The **keys()** method will return a list of all the keys in the dictionary.

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
x = Dict.values()  
print(x)
```



Python Dictionary

❑ Accessing Items

❑ keys()

The **keys()** method will return a list of all the keys in the dictionary.

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
x = Dict.keys()  
print(x)
```

❑ values()

The **values()** method will return a list of all the **values** in the dictionary.

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
x = Dict.values()  
print(x)
```



Python Dictionary

❑ Adding Items

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
Dict['DA'] = 21  
print(Dict)
```

❑ update() method

The **update()** method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

Ex:

```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
Dict.update({'Python': 'Django'})  
print(Dict)
```



Python Dictionary

❑ Removing Items

The **pop()** method removes the item with the specified key name

Ex:

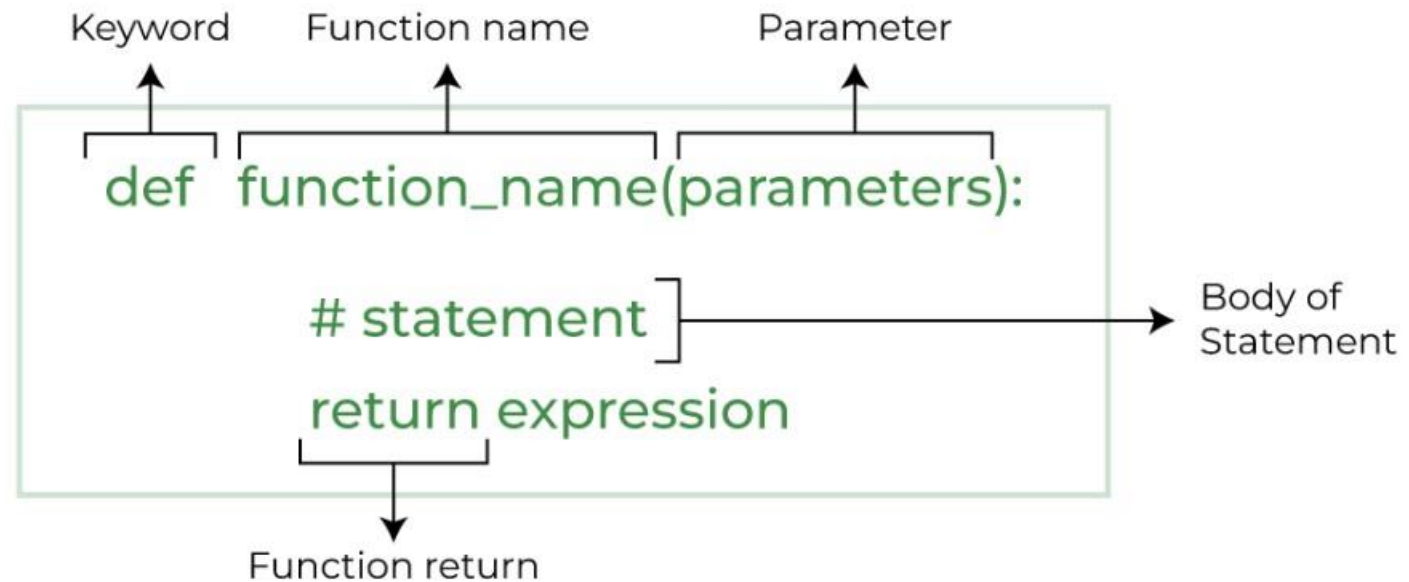
```
Dict = {'Python': 4, 'ML': 22, 'DL': 'AiQuest'}  
Dict.pop('ML')  
print(Dict)
```

Python Functions

Python Functions is a block of statements that return the specific task. It runs when it is called.

❑ In Python a function is defined using the **def** keyword.

❑ **Syntax:**



❑ Creating a Function:


Ex:

```
def first_function():  
    print('Welcome to python function lecture.')
```

❑ Calling a Function:

Ex:

```
def first_function():  
    print('Welcome to python function lecture.')
```

 `first_function()`

Python OOP

Object-oriented programming (OOP) is a method of structuring a program by bundling related properties and behaviors into individual objects.

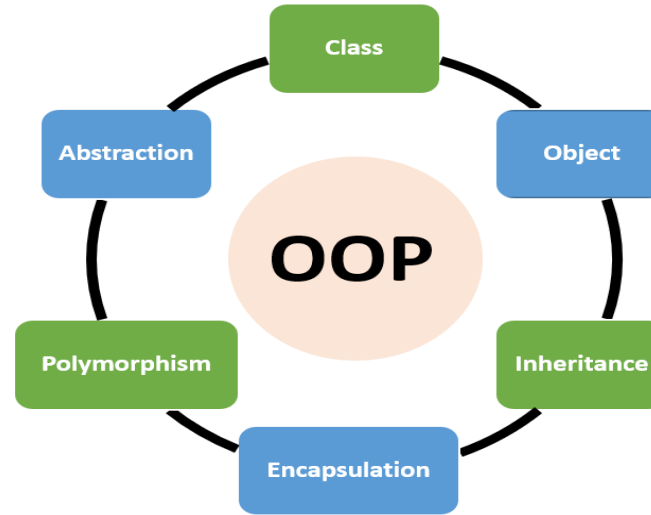
Ex:

Index	Object-oriented Programming	Procedural Programming
1.	Object-oriented programming is the problem-solving approach and used where computation is done by using objects.	Procedural programming uses a list of instructions to do computation step by step.
2.	It makes the development and maintenance easier.	In procedural programming, It is not easy to maintain the codes when the project becomes lengthy.
3.	It simulates the real world entity. So real-world problems can be easily solved through oops.	It doesn't simulate the real world. It works on step by step instructions divided into small parts called functions.
4.	It provides data hiding. So it is more secure than procedural languages. You cannot access private data from anywhere.	Procedural language doesn't provide any proper way for data binding, so it is less secure.

Python OOP

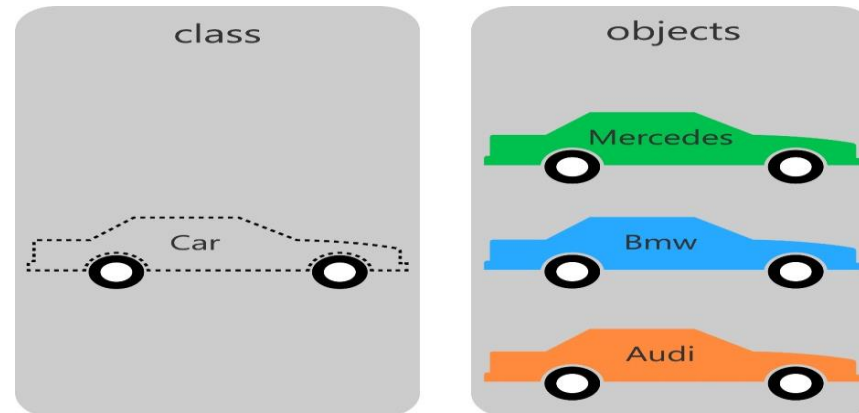
❑ Main concept of OOP:

Ex:

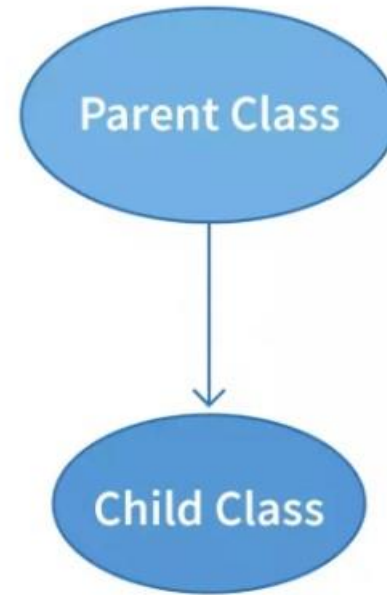


❑ Class & Object:

Ex:



Python Inheritance



Inheritance allows us to define a class that inherits all the methods and properties from another class.

- ❑ **Parent class** is the class being inherited from, also called base class.
- ❑ **Child class** is the class that inherits from another class, also called derived class.