

Classes **A**nd **O**bjects

Team **E**mertxe



Creation of Class



Creation of Class

General Format

- Class is a model or plan to create the objects
- Class contains,
 - **Attributes**: Represented by variables
 - **Actions** : Performed on methods
- Syntax of defining the class,

Syntax	Example
<pre>class Classname(object): """docstrings""" Attributes def __init__(self): def method1(): def method2():</pre>	<pre>class Student: """The below block defines attributes""" def __init__(self): self.name = "Ram" self.age = 21 self.marks = 89.75 """The below block defines a method""" def putdata(self): print("Name: ", self.name) print("Age: ", self.age) print("Marks: ", self.marks)</pre>

Creation of Class Program

#To define the Student class and create an Object to it.

#Class Definition

class Student:

#Special method called constructor

def __init__(self):

self.name = "Ram"

self.age = 21

self.marks = 75.90

#This is an instance method

def putdata(self):

print("Name: ", self.name)

print("Age: ", self.age)

print("Marks: ", self.marks)

#Create an instance to the student class

s = Student()

#Call the method using an Object

s.putdata()

The **Self** Variable



The **Self** Variable

- 'Self' is the default variable that contains the memory address of the instance of the current class

```
s1 = Student()
```

- `s1` contains the memory address of the instance
- This memory address is internally and by default passed to 'self' variable

Usage-1:

```
def __init__(self):
```

- The 'self' variable is used as first parameter in the constructor

Usage-2:

```
def putdata(self):
```

- The 'self' variable is used as first parameter in the instance methods

Constructor



Constructor

Constructor with **NO** parameter

- Constructors are used to create and initialize the 'Instance Variables'

Example

```
def __init__(self):  
    self.name = "Ram"  
    self.marks = 99
```

- Constructor will be called only once i.e at the time of creating the objects
- `s = Student()`

Constructor

Constructor with parameter



Example	<pre>def __init__(self, n = "", m = 0): self.name = n self.marks = m</pre>
Instance-1	<pre>s = Student()</pre> <p>Will initialize the instance variables with default parameters</p>
Instance-2	<pre>s = Student("Ram", 99)</pre> <p>Will initialize the instance variables with parameters passed</p>

Constructor Program

#To create Student class with a constructor having more than one parameter

```
class Student:
    #Constructor definition
    def __init__(self, n = "", m = 0):
        self.name = n
        self.marks = m

    #Instance method
    def putdata(self):
        print("Name: ", self.name)
        print("Marks: ", self.marks)

#Constructor called without any parameters
s = Student()
s.putdata()

#Constructor called with parameters
s = Student("Ram", 99)
s.putdata()
```



Types of **V**ariables

Types Of Variables

- Instance variables
- Class / Static variables

Types Of Variables

Instance Variables

- Variables whose separate copy is created for every instance/object
- These are defined and init using the constructor with 'self' parameter
- Accessing the instance variables from outside the class,
 - `instancename.variable`

```
class Sample:
    def __init__(self):
        self.x = 10

    def modify(self):
        self.x += 1
```

```
#Create an objects
s1 = Sample()
s2 = Sample()

print("s1.x: ", s1.x)
print("s2.x: ", s2.x)

s1.modify()
print("s1.x: ", s1.x)
print("s2.x: ", s2.x)
```

Types Of Variables

Class Variables

- Single copy is created for all instances
- Accessing class vars are possible only by 'class methods'
- Accessing class vars from outside the class,
 - `classname.variable`

```
class Sample:
    #Define class var here
    x = 10

    @classmethod
    def modify(cls):
        cls.x += 1
```

```
#Create an objects
s1 = Sample()
s2 = Sample()

print("s1.x: ", s1.x)
print("s2.x: ", s2.x)

s1.modify()
print("s1.x: ", s1.x)
print("s2.x: ", s2.x)
```

Namespaces



Namespaces

Introduction

- Namespace represents the memory block where names are mapped/linked to objects
- Types:
 - Class namespace
 - - The names are mapped to class variables
 - Instance namespace
 - - The names are mapped to instance variables

Namespaces

Class Namespace

#To understand class namespace

#Create the class

```
class Student:  
    #Create class var  
    n = 10
```

#Access class var in class namespace

```
print(Student.n)
```

#Modify in class namespace

```
Student.n += 1
```

#Access class var in class namespace

```
print(Student.n)
```

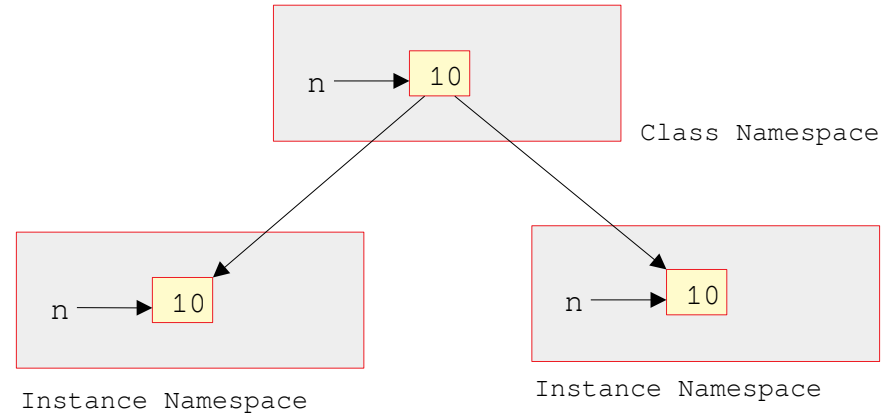
#Access class var in all instances

```
s1 = Student()  
s2 = Student()
```

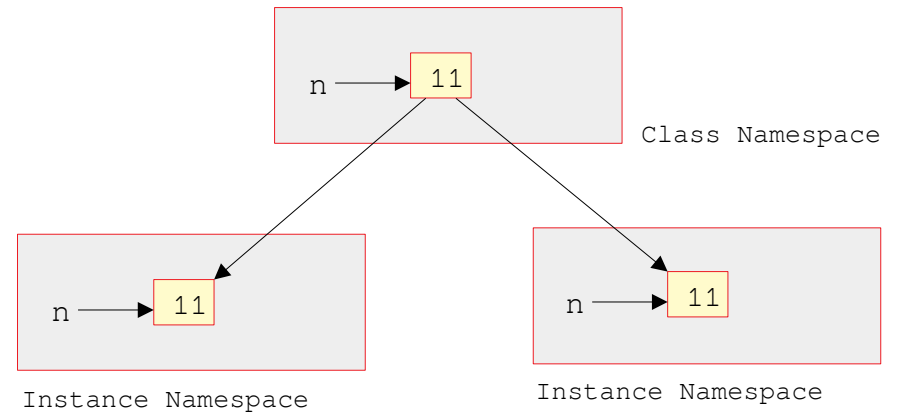
#Access class var in instance namespace

```
print("s1.n: ", s1.n)  
print("s2.n: ", s2.n)
```

Before modifyng class variable 'n'



After modifyng class variable 'n'



If class vars are modified in class namespace, then it reflects to all instances

Namespaces

Instance Namespace

#To understand class namespace

#Create the class

```
class Student:  
    #Create class var  
    n = 10
```

```
s1 = Student()  
s2 = Student()
```

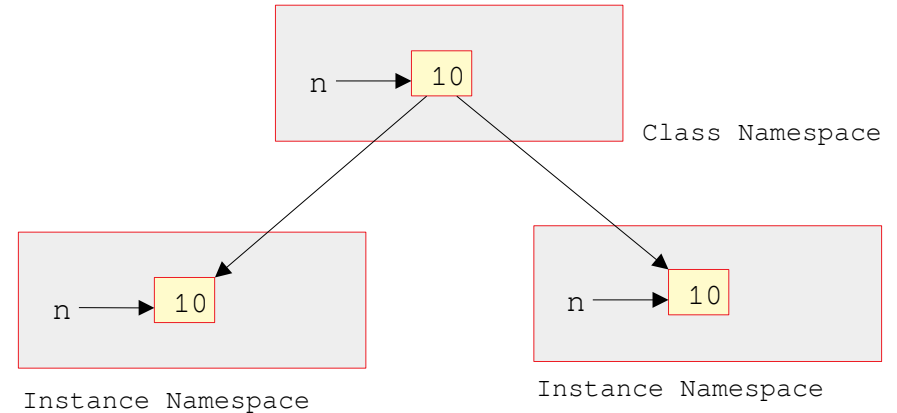
#Modify the class var in instance namespace

```
s1.n += 1
```

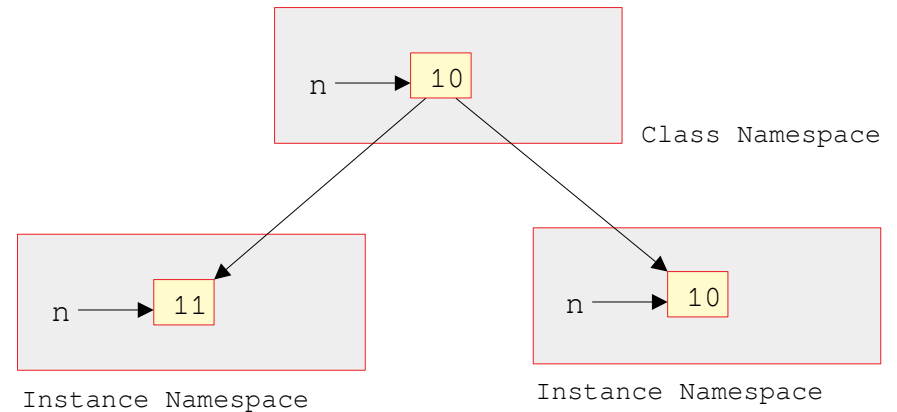
#Access class var in instance namespace

```
print("s1.n: ", s1.n)  
print("s2.n: ", s2.n)
```

Before modifyng class variable 'n'



After modifyng class variable 'n'



If class vars are modified in instance namespace, then it reflects only to that instance

Types of **M**ethods



Types of Methods

- **Types:**
 - Instance Methods
 - - Accessor
 - - Mutator
 - Class Methods
 - Static Methods

Types of Methods

Instance Methods

- Acts upon the instance variables of that class
- Invoked by `instance_name.method_name()`

#To understand the instance methods

```
class Student:
```

```
    #Constructor definition
```

```
    def __init__(self, n = "", m = 0):
        self.name = n
        self.marks = m
```

```
    #Instance method
```

```
    def putdata(self):
        print("Name: ", self.name)
        print("Marks: ", self.marks)
```

#Constructor called without any parameters

```
s = Student()
s.putdata()
```

#Constructor called with parameters

```
s = Student("Ram", 99)
s.putdata()
```

Types of Methods

Instance Methods: Accessor + Mutator

Accessor

- Methods just reads the instance variables, will not modify it
- Generally written in the form: `getXXXX()`
- Also called `getter` methods

Mutator

- Not only reads the data but also modifies it
- Generally written in the form: `setXXXX()`
- Also called `setter` methods

#To understand accessor and mutator

#Create the class

```
class Student:
```

#Define mutator

```
def setName(self, name):  
    self.name = name
```

#Define accessor

```
def getName(self):  
    return self.name
```

#Create an objects

```
s = Student()
```

#Set the name

```
s.setName("Ram")
```

#Print the name

```
print("Name: ", s.getName())
```

Types of Methods

Class Methods

- This methods acts on class level
- Acts on class variables only
- Written using `@classmethod` decorator
- First param is 'cls', followed by any params
- Accessed by `classname.method()`

#To understand the class methods

```
class Bird:
    #Define the class var here
    wings = 2

    #Define the class method
    @classmethod
    def fly(cls, name):
        print("{} flies with {} wings" . format(name, cls.wings))

#Call
Bird.fly("Sparrow")
Bird.fly("Pigeon")
```

Types of Methods

Static Methods

- Needed, when the processing is at the class level but we need not involve the class or instances
- Examples:
 - Setting the environmental variables
 - Counting the number of instances of the class
- Static methods are written using the decorator `@staticmethod`
- Static methods are called in the form `classname.method()`

`#To Understand static method`

```
class Sample:
    #Define class vars
    n = 0

    #Define the constructor
    def __init__(self):
        Sample.n = Sample.n + 1

    #Define the static method
    @staticmethod
    def putdata():
        print("No. of instances created: ", Sample.n)
```

`#Create 3 objects`

```
s1 = Sample()
s2 = Sample()
s3 = Sample()
```

`#Class static method`

```
Sample.putdata()
```


Passing **M**embers



Passing Members

- It is possible to pass the members(attributes / methods) of one class to another
- Example:

```
e = Emp()
```

- After creating the instance, pass this to another class 'Myclass'
- `Myclass.mymethod(e)`
 - mymethod is static

Passing Members

Example

#To understand how members of one class can be passed to another

#Define the class

```
class Emp:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def putdata(self):
        print("Name: ", self.name)
        print("Salary: ", self.salary)
```

#Define another class

```
class Myclass:
    @staticmethod
    def mymethod(e):
        e.salary += 1000
        e.putdata()
```

#Create Object

```
e = Emp("Ram", 20000)
```

#Call static method of Myclass and pass e

```
Myclass.mymethod(e)
```

Passing Members

Exercise

1. To calculate the power value of a number with the help of a static method

Inner Class



Inner Class

Introduction

- Creating class B inside Class A is called nested class or Inner class
- Example:

Person's Data like,

- Name: Single value
- Age: Single Value
- DoB: Multiple values, hence separate class is needed

Inner Class

Program: Version-1

#To understand inner class

```
class Person:
    def __init__(self):
        self.name = "Ram"
        self.db = self.Dob()

    def display(self):
        print("Name: ", self.name)

#Define an inner class
class Dob:
    def __init__(self):
        self.dd = 10
        self.mm = 2
        self.yy = 2002

    def display(self):
        print("DoB: {}/{}/{}" . format(self.dd,
self.mm, self.yy))
```

#Creating Object

```
p = Person()
p.display()
```

#Create inner class object

```
i = p.db
i.display()
```

Inner Class

Program: Version-2

#To understand inner class

```
class Person:
    def __init__(self):
        self.name = "Ram"
        self.db = self.Dob()

    def display(self):
        print("Name: ", self.name)

#Define an inner class
class Dob:
    def __init__(self):
        self.dd = 10
        self.mm = 2
        self.yy = 2002

    def display(self):
        print("DoB: {}/{}/{}".format(self.dd,
self.mm, self.yy))
```

#Creating Object

```
p = Person()
p.display()
```

#Create inner class object

```
i = Person().Dob()
i.display()
```


THANK YOU