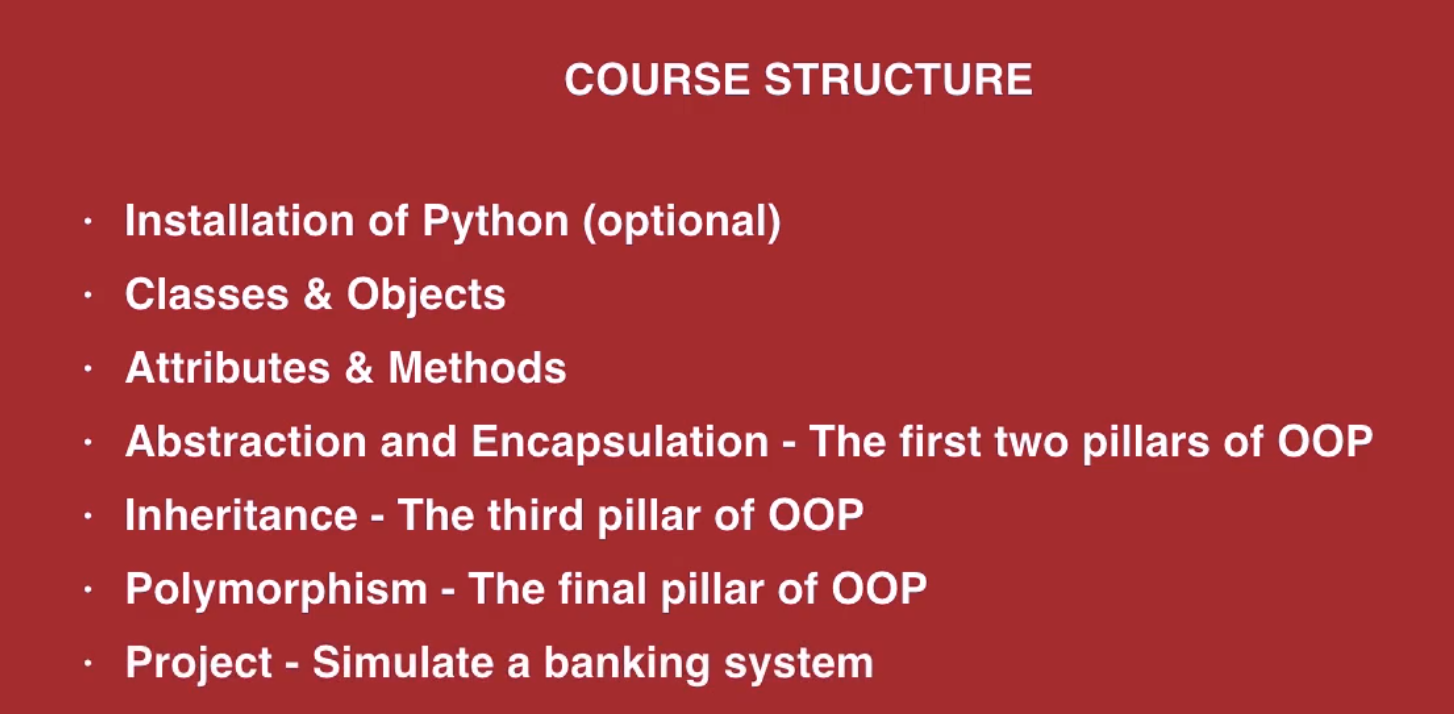
**Python OOP:**

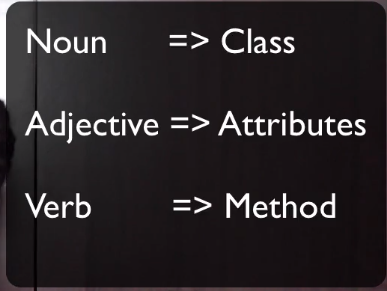


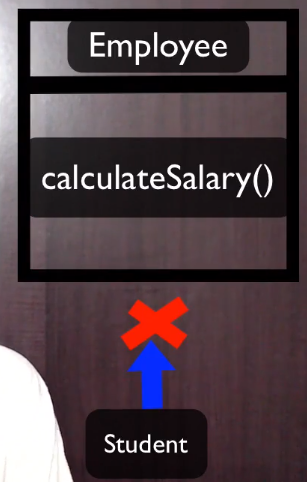
Install via python.org

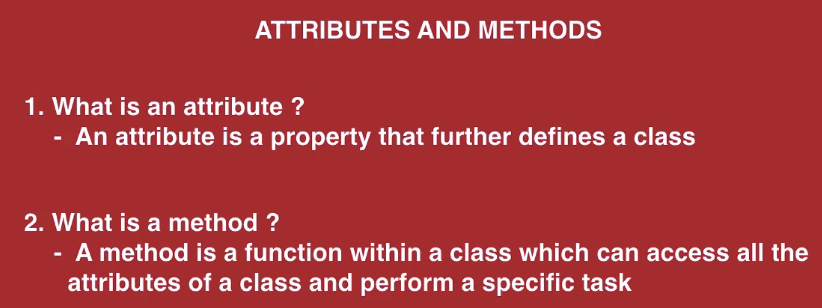
**Classes & Objects:**

**Class:** logical collection of attributes and methods

**Object:** instance of Class

****

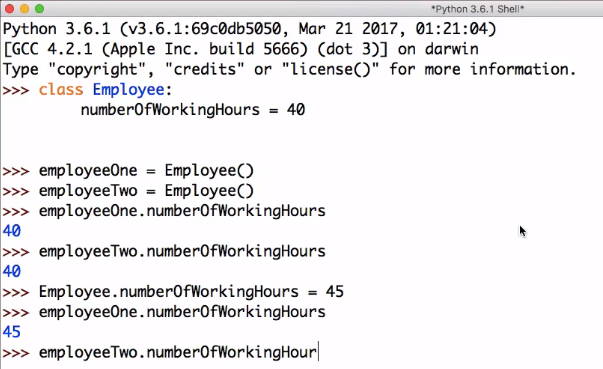
****

****

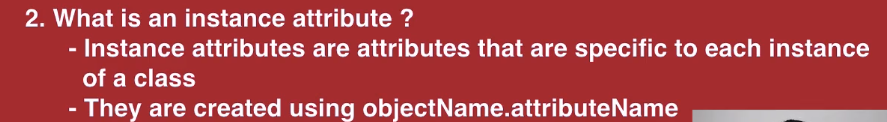
**Class attributes:**

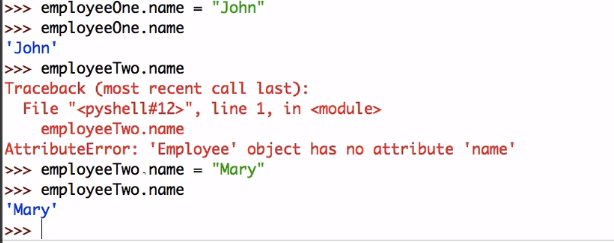
**Common to all class objects**

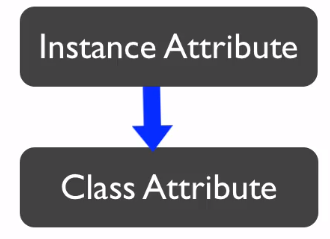
****

****

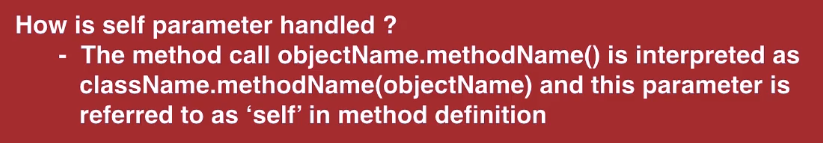
**Instance attributes:**

****

****

****

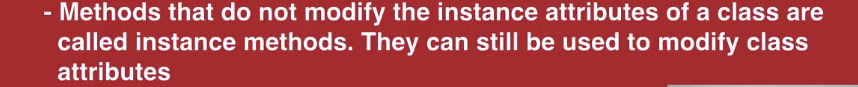
**Self parameter:**

****

**Static Methods and Instance MethodsSection 4, Lecture 9**

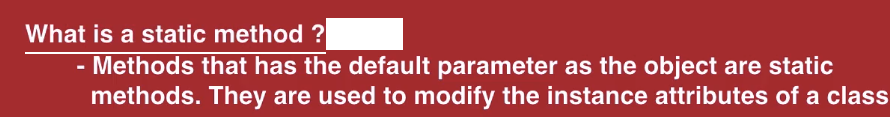
**Static Methods and Instance Methods:**

****

****

**@staticmethod**

* **Not using ‘**self’ **variable**



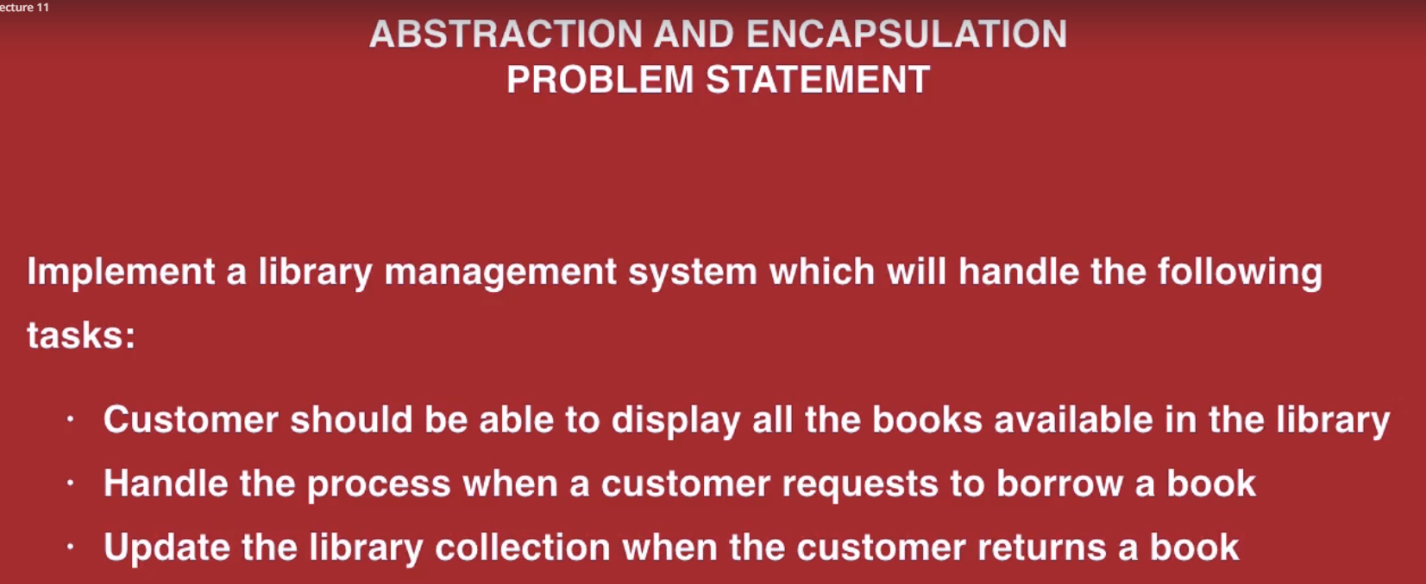
class Employee:  
 def employee\_details(self):  
 self.name = **"Ben"** @staticmethod  
 def welcome\_message():  
 print(**"Welcome to our Organization!"**)  
  
  
employee = Employee()  
employee.employee\_details()  
print(employee.name)  
employee.welcome\_message()  
#Employee.welcome\_message()

**init() method - Create a fully initialized object**

****

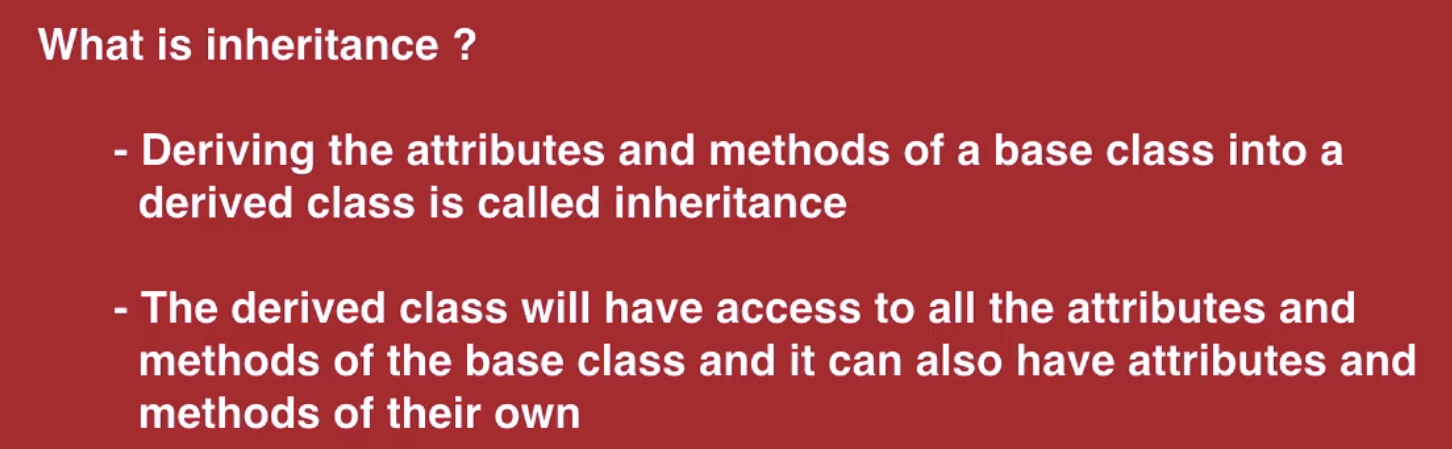
class Employee:  
  
 def \_\_init\_\_(self, name=**"Mark"**):  
 self.\_name = name  
  
 def display\_employee\_details(self):  
 print(self.\_name)  
  
employee= Employee()  
employee.display\_employee\_details()  
  
employeeTwo = Employee(**"Milson"**)  
employeeTwo.display\_employee\_details()

**Abstraction and Encapsulation:**

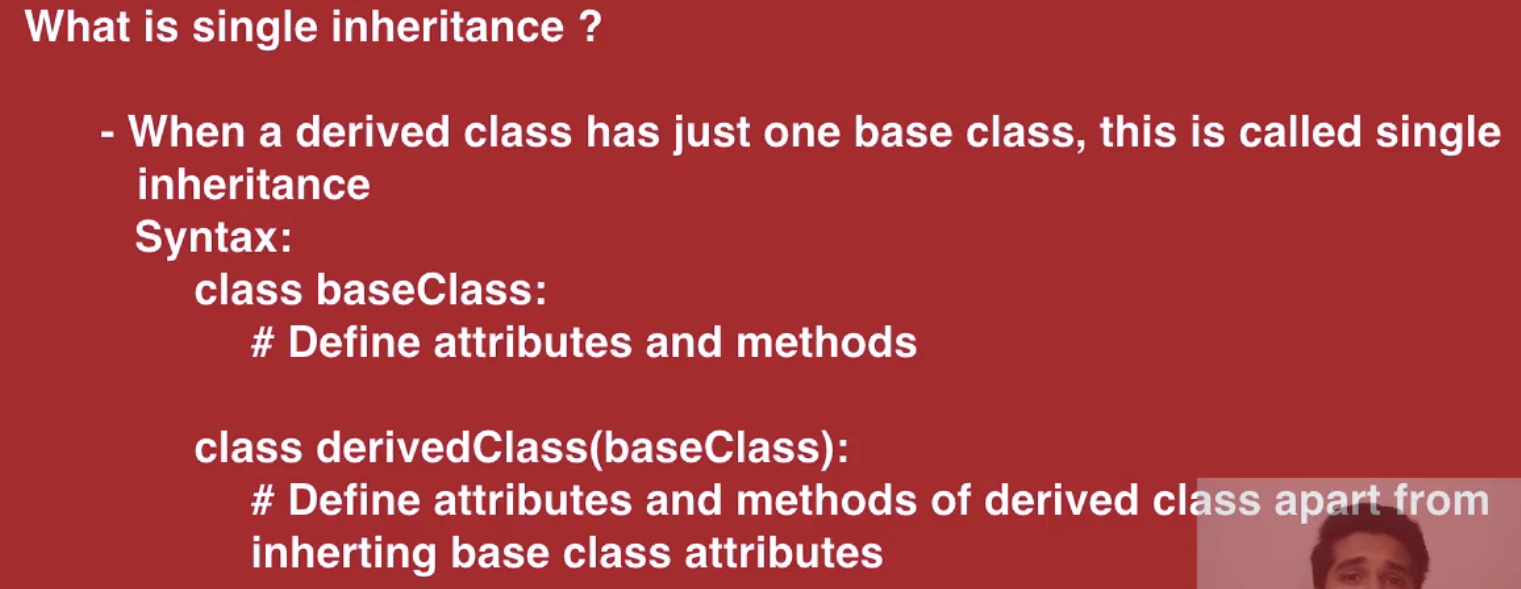
****

# Class => Library  
# layers of abstraction => display available books, to lend a book, to add a book  
  
# Class => Customer  
# Layers of abstraction => request for a book, return a book  
  
class Library(object):  
  
 def \_\_init\_\_(self, available\_books):  
 self.available\_books = available\_books  
  
 def display\_available\_books(self):  
 print()  
 print(**"Available Books: "**)  
 for book in self.available\_books:  
 print(book)  
 print()  
  
 def lend\_book(self, requested\_book):  
 if requested\_book in self.available\_books:  
 print(**"You have now borrowed the book"**)  
 self.available\_books.remove(requested\_book)  
 else:  
 print(**"Sorry! the book is not available in our list."**)  
  
 def add\_book(self, returned\_book):  
 self.available\_books.append(returned\_book)  
 print(**"You have returned the book. Thank you!!"**)  
  
  
class Customer(object):  
 def request\_book(self):  
 print(**"Enter the name of a book you would like to borrow: "**)  
 self.book = input()  
 return self.book  
  
 def return\_book(self):  
 print()  
 print(**"Enter the name of the book which you are returning: "**)  
 self.book = input()  
 return self.book  
  
  
library = Library([**'Think and Grow Rich'**, **'Who Will Cry When You Die'**, **'For One More Day'**])  
customer = Customer()  
  
while True:  
 print(**"Enter 1 to display the available books"**)  
 print(**"Enter 2 to request a book"**)  
 print(**"Enter 3 to return a book"**)  
 print(**"Enter 4 to exit"**)  
 user\_choice = int(input())  
  
 if user\_choice is 1:  
 library.display\_available\_books()  
 elif user\_choice is 2:  
 requested\_book = customer.request\_book()  
 library.lend\_book(requested\_book)  
 elif user\_choice is 3:  
 returned\_book = customer.request\_book()  
 library.add\_book(returned\_book)  
 elif user\_choice is 4:  
 quit()  
 else:  
 quit()

**Inheritance:**

****

1. **Single Inheritance:**



class Apple: # Base Class  
 manufacture = **'Apple Inc.'** website = **'http://www.apple.com'** def contact\_detail(self):  
 print(**"To contact us, log on to "**, self.website)  
  
  
class MacBook(Apple): # Derived Class  
 def \_\_init\_\_(self):  
 self.year\_of\_manufacture = 2017  
  
 def manufacture\_detail(self):  
 print(**"This MacBook was manufactured in the year {} by {}"**.format(self.year\_of\_manufacture, self.manufacture))  
  
  
mac\_book = MacBook()  
mac\_book.manufacture\_detail()  
mac\_book.contact\_detail()

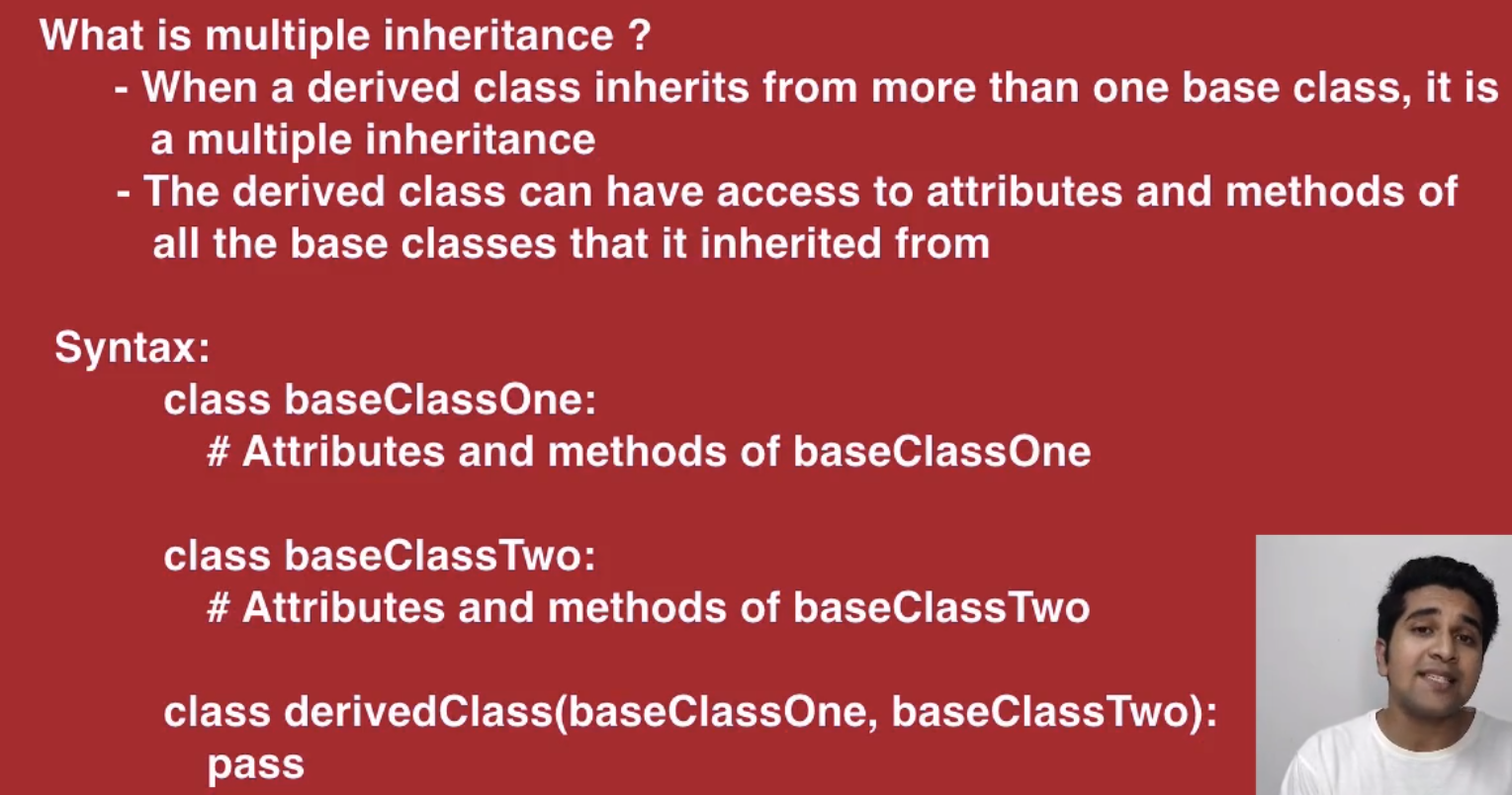
**output:**

***(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python singleinheritance.py***

***This MacBook was manufactured in the year 2017 by Apple Inc.***

***To contact us, log on to http://www.apple.com***

1. **Multiple Inheritance:**

****

class OperatingSystem: # Base Class  
 multitasking = True  
 name = **'MAC OS'**class Apple: # Base Class  
 website = **'http://www.apple.com'** name = **'Apple'**class MacBook(OperatingSystem, Apple): # Derived Class  
 def \_\_init\_\_(self):  
 if self.multitasking is True:  
 print(**"This is a multi tasking system. Visit {} for more detail"**.format(self.website))  
 print(**'Name: '**, self.name) # the order of inheritance in derived class will determine this conflicting  
 # name's value First declared base class has high precedence  
  
mac\_book = MacBook()

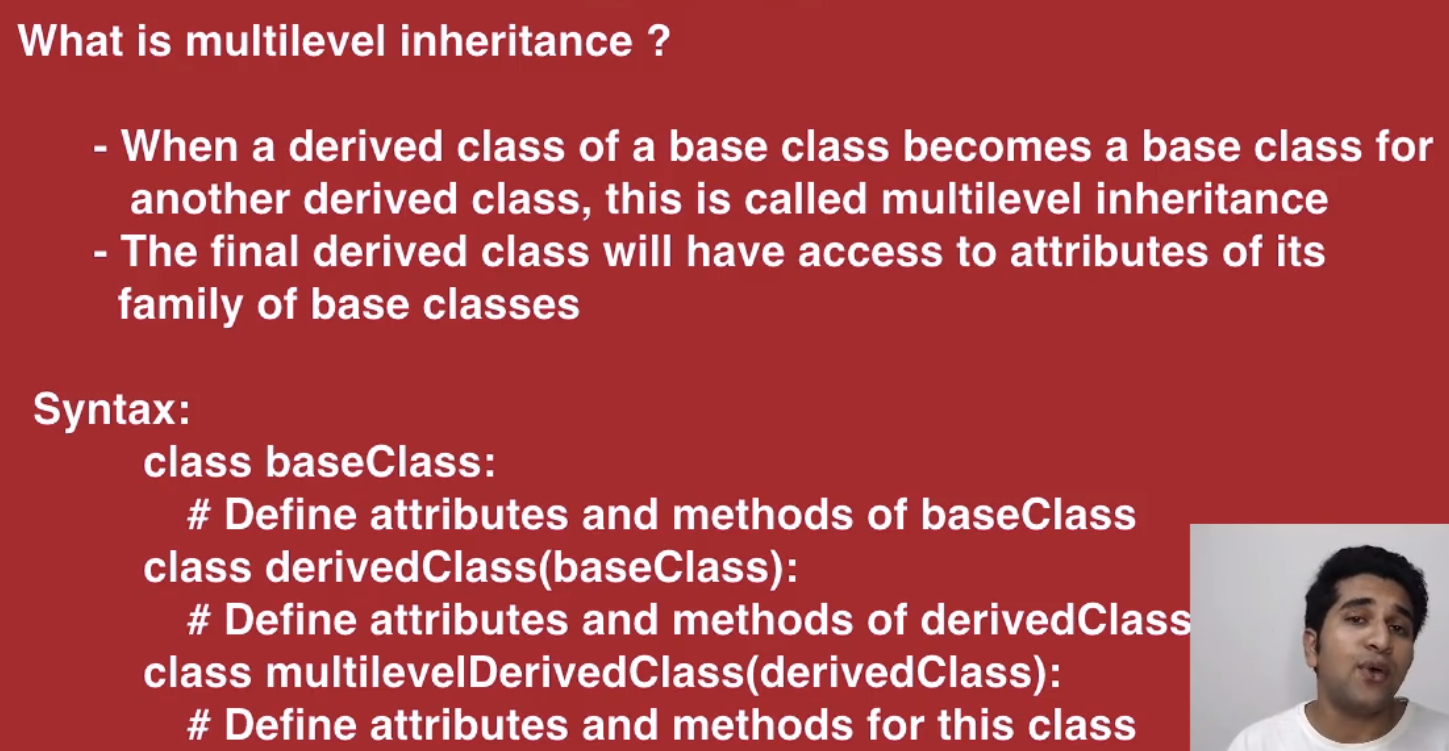
**output:**

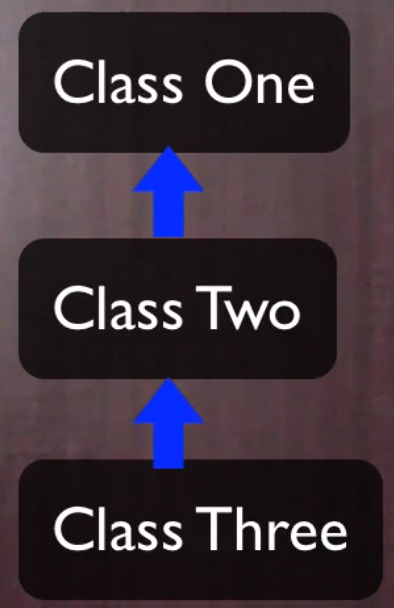
***(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python multipleinheritance.py***

***This is a multi tasking system. Visit http://www.apple.com for more detail***

***Name: MAC OS***

1. **Multi-level Inheritance:**

****

****

**\*\*\* Diamond Shape Problem in Multiple Inheritance**

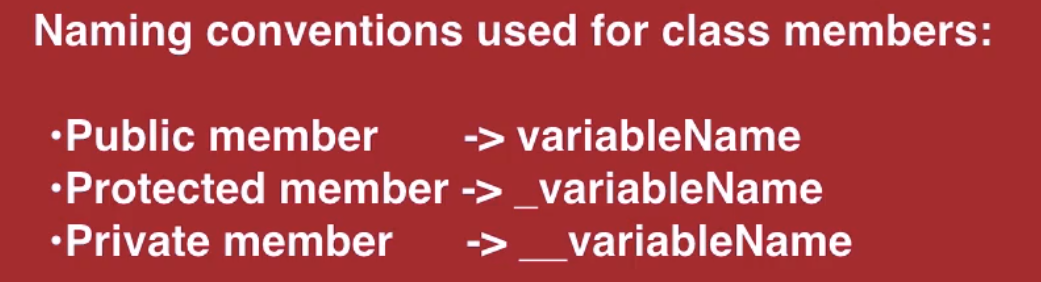
class MusicalInstrument:  
 no\_of\_major\_keys = 12  
  
  
class StringInsturment(MusicalInstrument):  
 type\_of\_wood = **'Tonewood'**class Guitar(StringInsturment):  
 def \_\_init\_\_(self):  
 self.no\_of\_strings = 6  
 print(  
 **'This guitar consists of {} strings. It is made up of {} and it can play {} keys.'**.format(self.no\_of\_strings, self.type\_of\_wood, self.no\_of\_major\_keys))  
  
  
guitar = Guitar()

***(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python multilevelinheritance.py***

***This guitar consists of 6 strings. It is made up of Tonewood and it can play 12 keys.***

**Access modifiers:**

***Public: (class and derived class also outside accessible), Protecte: (only within class and derived class) and Private (only within defined class) naming Conventions***

******

# Public => memberName  
# Protected => \_memeberName  
# Private => \_\_memberName  
  
  
class Car:  
 no\_of\_wheels = 4  
 \_color = **'Black'** \_\_year\_of\_manufacture = 2019 # \_car\_\_year\_of\_manufacture  
  
 def \_\_init\_\_(self):  
 self.\_\_year\_of\_manufacture = 2018  
  
class BMW(Car):  
 def \_\_init\_\_(self):  
 print(**"Protected attribute \_color: "**, self.\_color)  
  
car = Car()  
print(**"Public attribute no\_of\_wheels: "**, car.no\_of\_wheels)   
  
bmw = BMW()

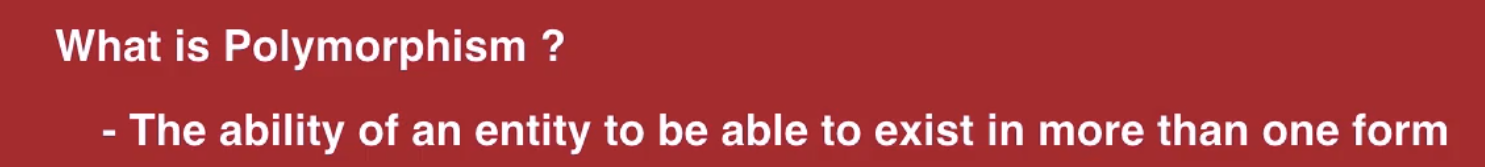
Output:

*(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python accessSpecifiers.py*

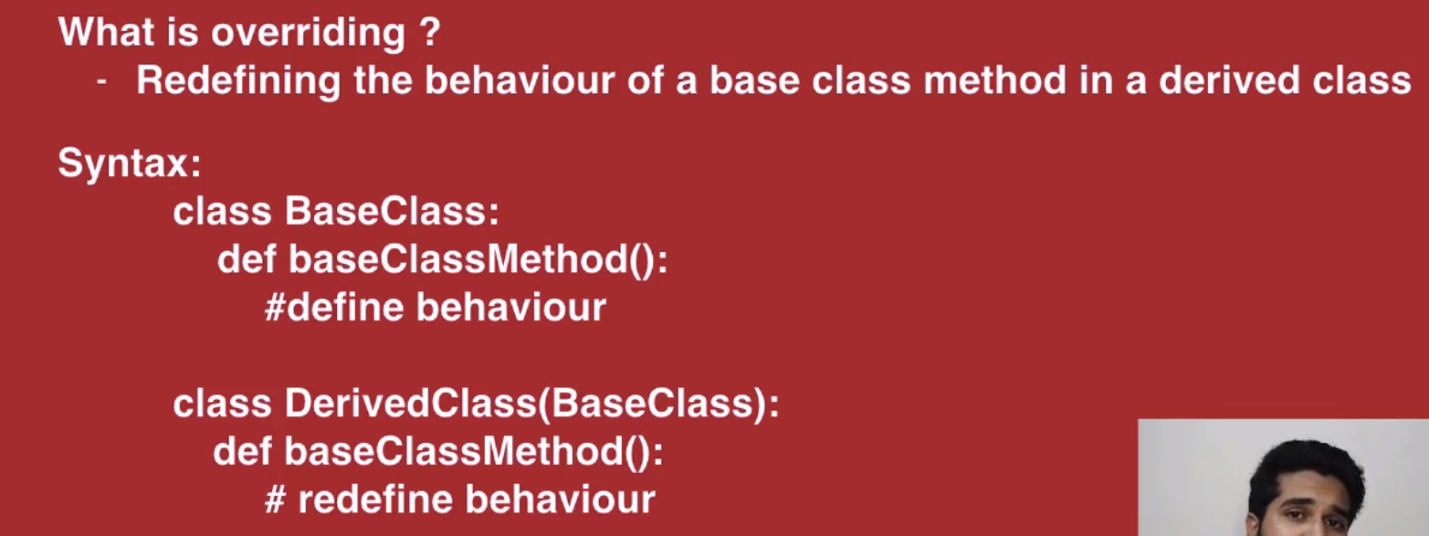
*Public attribute no\_of\_wheels: 4*

*Protected attribute \_color: Black*

**Polymorphism:**

****

**Overriding:**

****

class Employee:  
 def set\_no\_of\_working\_hours(self):  
 self.no\_of\_working\_hours = 40  
  
 def get\_no\_of\_working\_hours(self):  
 print(self.no\_of\_working\_hours)  
  
class Trainee(Employee):  
 def set\_no\_of\_working\_hours(self):  
 self.no\_of\_working\_hours = 45  
  
 def reset\_no\_of\_working\_hours(self):  
 super().set\_no\_of\_working\_hours()  
  
  
  
employee = Employee()  
employee.set\_no\_of\_working\_hours()  
print(**"Number of working hours of employee:"**, end=**' '**)  
employee.get\_no\_of\_working\_hours()  
  
trainee = Trainee()  
trainee.set\_no\_of\_working\_hours()  
print(**"Number of working hours of trainee:"**, end=**' '**)  
trainee.get\_no\_of\_working\_hours()  
trainee.reset\_no\_of\_working\_hours()  
print(**"Number of working hours of trainee after reset:"**, end=**' '**)  
trainee.get\_no\_of\_working\_hours()

**Output:**

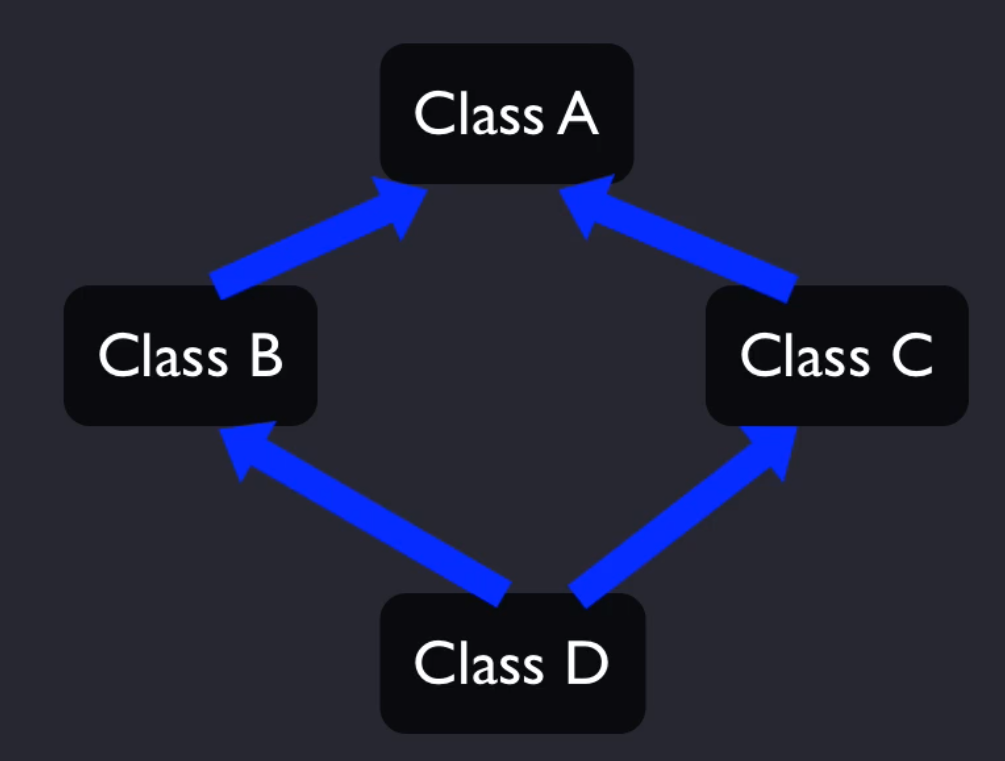
*(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python overriding.py*

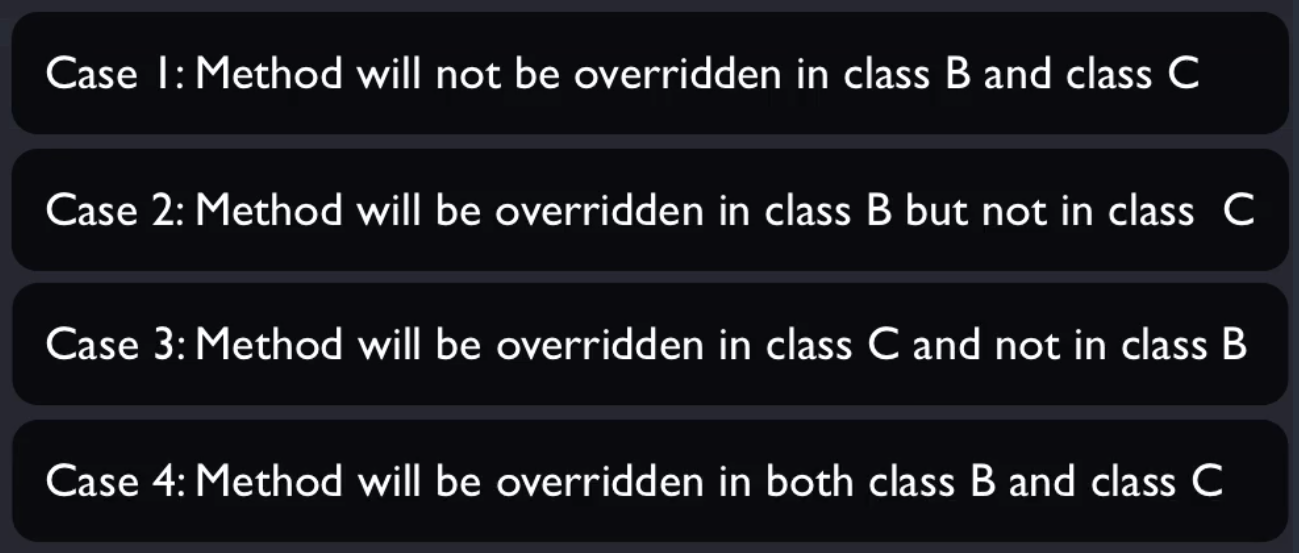
*Number of working hours of employee: 40*

*Number of working hours of trainee: 45*

*Number of working hours of trainee after reset: 40*

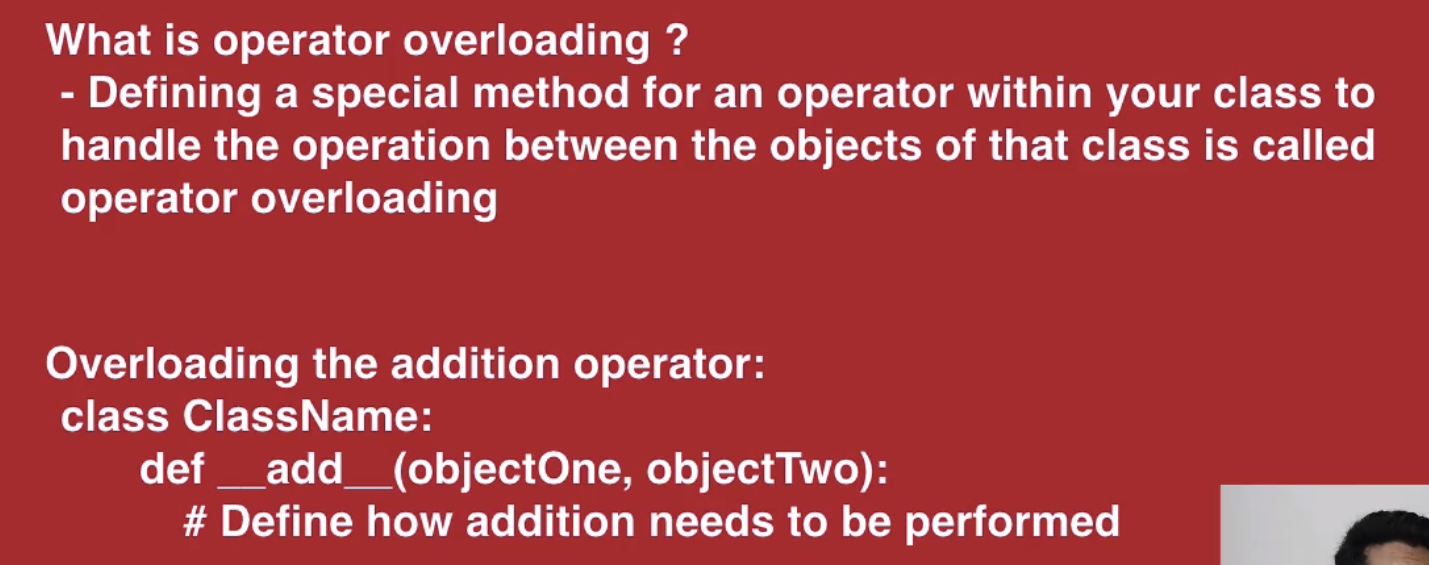
**Diamond Shape Problem in Multiple Inheritance:**

****

****

class A:  
 def method(self):  
 print(**"This method belongs to class A."**)  
 pass  
  
  
class B(A):  
 def method(self):  
 print(**"This method belongs to class B."**)  
 pass  
  
  
class C(A):  
 def method(self):  
 print(**"This method belongs to class C."**)  
 pass  
  
  
class D(B, C):# order of declaration of base class matters!  
 pass  
  
  
d = D()  
d.method()

**Overloading an Operator:**

****

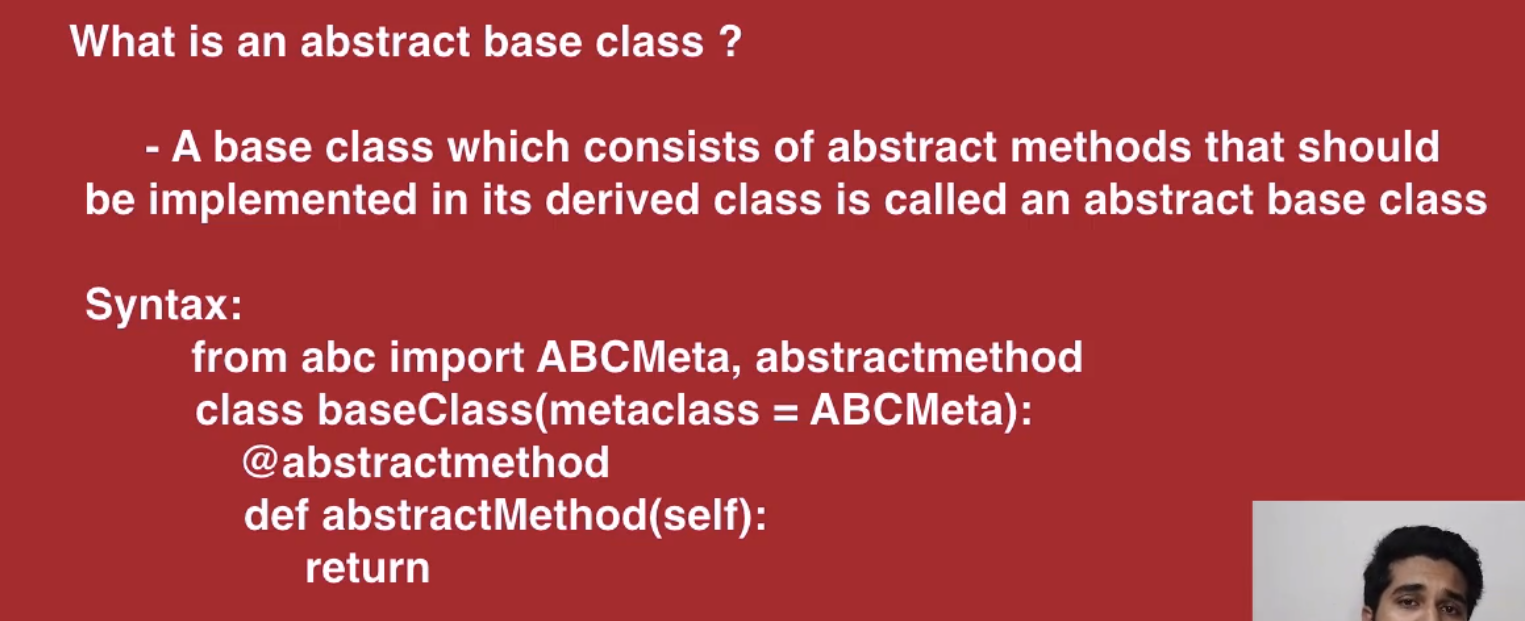
class Square:  
 def \_\_init\_\_(self, side):  
 self.side = side  
  
 def \_\_add\_\_(square\_one, square\_two): # special name for operator overloading method for +   
 return (4 \* square\_one.side) + (4 \* square\_two.side)  
  
  
  
square\_one = Square(5) # 5\*4=20  
square\_two = Square(10) # 10\*4=40  
  
print(**"Sum of sides of both squares: "**, square\_one + square\_two)

**Output:**

*(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python overloading.py*

*Sum of sides of both squares: 60*

**Abstract Base Class:**

****

from abc import ABCMeta, abstractmethod  
  
  
class Shape(metaclass=ABCMeta):  
  
 @abstractmethod  
 def area(self):  
 return 0  
  
  
class Square(Shape):  
 side = 4  
  
 def area(self):  
 print(**"Area of Square: "**, self.side \* self.side)  
  
  
class Rectangle(Shape):  
 length = 10  
 width = 5  
  
 def area(self):  
 print(**"Area of Rectangle: "**, self.length \* self.width)  
  
  
square = Square()  
rectangle = Rectangle()  
  
square.area()  
rectangle.area()  
# shape = Shape() # Can't instantiate abstract class Shape with abstract methods area

**Output:**

*(venv) C:\Users\Sarath Sathiraju\PycharmProjects\PythonOOPLearning>python abstractbaseclass.py*

*Area of Square: 16*

*Area of Rectangle: 50*

**Project Work:**

****

from abc import abstractmethod, ABCMeta  
from random import randint  
  
  
class Account(metaclass=ABCMeta):  
  
 @abstractmethod  
 def create\_account(self, name, initial\_deposit):  
 return 0  
  
 @abstractmethod  
 def authenticate(self, name, account\_no):  
 return 0  
  
 @abstractmethod  
 def withdraw(self, withdrawal\_amount):  
 return 0  
  
 @abstractmethod  
 def deposit(self, deposit\_amount):  
 return 0  
  
 @abstractmethod  
 def display\_balance(self):  
 return 0  
  
  
class SavingsAccount(Account):  
  
 def \_\_init\_\_(self):  
 # key => account\_no; [key][0] => name; [key][1] => balance  
 self.saving\_accounts\_list = {}  
  
 def create\_account(self, name, initial\_deposit):  
 self.account\_no = randint(10000, 99999)  
 self.saving\_accounts\_list[self.account\_no] = [name, initial\_deposit]  
 print(**'Account creation was successful. You new account number is: '**, self.account\_no)  
  
 def authenticate(self, name, account\_no):  
 if account\_no in self.saving\_accounts\_list.keys():  
 if self.saving\_accounts\_list[account\_no][0] == name:  
 print(**'Authentication Successful!'**)  
 self.account\_no = account\_no  
 return True  
 else:  
 print(**'Authentication Failed!!!'**)  
 return False  
 else:  
 print(**'Authentication Failed!!!'**)  
 return False  
  
 def withdraw(self, withdrawal\_amount):  
 if withdrawal\_amount > self.saving\_accounts\_list[self.account\_no][1]:  
 print(**'Insufficient balance!'**)  
 else:  
 self.saving\_accounts\_list[self.account\_no][1] -= withdrawal\_amount  
 print(**'Withdrawal was successful.'**)  
 self.display\_balance()  
  
 def deposit(self, deposit\_amount):  
 self.saving\_accounts\_list[self.account\_no][1] += deposit\_amount  
 print(**'Deposit was successful.'**)  
 self.display\_balance()  
  
 def display\_balance(self):  
 print(**'Available balance: '**, self.saving\_accounts\_list[self.account\_no][1])  
  
  
saving\_account = SavingsAccount()  
  
while True:  
 print(**'Enter 1 to create a new account'**)  
 print(**'Enter 2 to access an existing account'**)  
 print(**'Enter 3 to exit'**)  
  
 user\_choice = int(input())  
 if user\_choice is 1: # create a new account  
 print(**'Enter your name: '**)  
 name = input()  
 print(**'Enter your initial deposit: '**)  
 initial\_deposit = int(input())  
 saving\_account.create\_account(name, initial\_deposit)  
 elif user\_choice is 2: # access the account  
 print(**'Enter your name: '**)  
 name = input()  
 print(**'Enter your account no: '**)  
 account\_no = int(input())  
  
 auth\_status = saving\_account.authenticate(name, account\_no)  
 if auth\_status:  
 while True:  
 print(**'Enter 1 to Withdraw'**)  
 print(**'Enter 2 to Deposit'**)  
 print(**'Enter 3 to see your balance'**)  
 print(**'Enter 4 to go back to the previous menu'**)  
  
 user\_choice = int(input())  
  
 if user\_choice is 1: # Withdraw  
 print(**'Enter your withdrawal amount: '**)  
 withdrawal\_amount = int(input())  
 saving\_account.withdraw(withdrawal\_amount)  
 elif user\_choice is 2: # Deposit  
 print(**'Enter your deposit amount: '**)  
 deposit\_amount = int(input())  
 saving\_account.deposit(deposit\_amount)  
 elif user\_choice is 3: # View Balance  
 saving\_account.display\_balance()  
 elif user\_choice is 4: # Return back  
 break  
  
 elif user\_choice is 3: # Exit  
 quit()