DAY-7 EVENING ASSESSMENT

1. class Order:  
   def \_\_init\_\_(self, order\_id):  
       self.order\_id = order\_id  
  
   def process\_payment(self):  
       print(f"Processing payment for Order #{self.order\_id}")  
  
class OnlineOrder(Order):  
   def \_\_init\_\_(self, order\_id, email):  
       super().\_\_init\_\_(order\_id)  # Calls Order's \_\_init\_\_ to set order\_id  
       self.email = email          # Adds email attribute specific to OnlineOrder  
  
   def process\_payment(self):  
       super().process\_payment()  # Calls Order's process\_payment()  
       print(f"Sending confirmation email to {self.email}")  
  
# Creating an OnlineOrder object  
order = OnlineOrder(101, "customer@example.com")  
order.process\_payment()

**Online order is the child class and order is the base class.Onlineorder inherits from order.**

**Onlineorder overrides process\_payment() method and adds extra behaviour with the help of super keyword.**

**o/p: Processing payment for Order #101  
Sending confirmation email to customer@example.com**

**2. Employee and Manager**  
class Employee:  
   def \_\_init\_\_(self, name, salary):  
       self.name = name  
       self.salary = salary  
  
   def display(self):  
       print(f"Name: {self.name}, Salary: ₹{self.salary}")  
  
class Manager(Employee):  
   def \_\_init\_\_(self, name, salary, department):  
       super().\_\_init\_\_(name, salary)  
       self.department = department  
  
   def display(self):  
       super().display()  
       print(f"Department: {self.department}")  
  
m = Manager("Shaik", 90000, "IT")  
m.display()

**🡪Employee is the base class(applied to all employees)**

**🡪Manager is the derived class(is one of the employees),method overriding happens here adding more info about manager.**

**🡪when employee has a name and salary while manager has name,salary as well as department.**

o/p: Name: Shaik, Salary: ₹90000  
Department: IT

**3. Vehicle and Car**class Vehicle:  
   def start(self):  
       print("Vehicle started")  
  
class Car(Vehicle):  
   def start(self):  
       super().start()  
       print("Car is ready to go")  
  
c = Car()  
c.start()

🡪Vehicle-base class which is applicable to all vehicles

🡪car is derived class which includes vehicle properties and super keyword helps to inherit the start method from the base class vehicle .

🡪here the output is” Vehicle Started Car is ready to go “

**4. User Login System**  
class User:  
   def \_\_init\_\_(self, username):  
       self.username = username  
  
   def login(self):  
       print(f"{self.username} logged in")  
  
class Admin(User):  
   def login(self):  
       super().login()  
       print(f"{self.username} has admin privileges")  
  
a = Admin("admin\_user")  
a.login()

🡪User-base class,when user logs in it says user logged in Admin—derived class,when admin logs in it says user logged in and says user has admin privileges.

🡪here in the code the username is admin\_user

It prints:

admin\_user logged in #from base class using super

admin\_user has admin privileges #overrides base class.

**5. Shape and Circle**  
class Shape:  
   def \_\_init\_\_(self):  
       print("This is a shape")  
  
   def area(self):  
       print("Area formula not defined")  
  
class Circle(Shape):  
   def \_\_init\_\_(self, radius):  
       super().\_\_init\_\_()  
       self.radius = radius  
  
   def area(self):  
       super().area()  
       print("Circle Area:", 3.14 \* self.radius \* self.radius)  
  
c = Circle(5)  
c.area()

 🡪Shape-base class, Circle-derived class

🡪when circle object is created it runs the shape constructor and stores the radius value

🡪then calls the shape’s area method ,overrides the shapes area method .

🡪output is

This is a shape

Area formula not defined

Circle Area:78.5

**6. Person and Student**  
class Person:  
   def \_\_init\_\_(self, name):  
       self.name = name  
  
   def show(self):  
       print(f"Name: {self.name}")  
  
class Student(Person):  
   def \_\_init\_\_(self, name, grade):  
       super().\_\_init\_\_(name)  
       self.grade = grade  
  
   def show(self):  
       super().show()  
       print(f"Grade: {self.grade}")  
  
s = Student("Ali", "A")  
s.show()

🡪Base class: Person, stores the name and prints name if show method in person class is called.

🡪derived class: Student ,stores name from person class (super) and grade ,when show method is called it prints name from person class as well as grade .

🡪Name:Ali

Grade: A

**7. BankAccount and SavingsAccount**  
class BankAccount:  
   def \_\_init\_\_(self, balance):  
       self.balance = balance  
  
   def show\_balance(self):  
       print(f"Balance: ₹{self.balance}")  
  
class SavingsAccount(BankAccount):  
   def \_\_init\_\_(self, balance, interest):  
       super().\_\_init\_\_(balance)  
       self.interest = interest  
  
   def show\_balance(self):  
       super().show\_balance()  
       print(f"Interest Rate: {self.interest}%")  
  
acc = SavingsAccount(10000, 5)  
acc.show\_balance()

🡪BankAccount – base class

🡪Savings Account- Derived class

🡪SavingsAccount.\_\_init\_\_() calls the parent constructor to store balance and also stores interest.

🡪SavingsAccount.show\_balance():  
• First shows balance using super().show\_balance()  
• Then prints the interest rate

o/p:Balance:10000

Interest Rate:5%

**8. Product and ElectronicProduct**  
class Product:  
   def \_\_init\_\_(self, name):  
       self.name = name  
  
   def details(self):  
       print(f"Product: {self.name}")  
  
class ElectronicProduct(Product):  
   def \_\_init\_\_(self, name, warranty):  
       super().\_\_init\_\_(name)  
       self.warranty = warranty  
  
   def details(self):  
       super().details()  
       print(f"Warranty: {self.warranty} years")  
  
p = ElectronicProduct("Laptop", 2)  
p.details()

🡪 Product stores the name and shows it using details().  
🡪ElectronicProduct adds a warranty and extends details() to show it.  
🡪p.details() prints the product name and warranty. 1. Product stores the name and shows it using details().

o/p:Product :Laptop

Warranty: 2 years

**9. Animal and Dog**  
class Animal:  
   def sound(self):  
       print("Animal sound")  
  
class Dog(Animal):  
   def sound(self):  
       super().sound()  
       print("Dog barks")  
  
d = Dog()  
d.sound()

🡪 Dog inherits features from Animal  
🡪Dog defines its own version of sound()  
🡪super()-Used to call the parent class (Animal) version of the method

o/p:Animal Sound

Dog barks

**10. Book and EBook**  
class Book:  
   def \_\_init\_\_(self, title):  
       self.title = title  
  
   def show(self):  
       print(f"Title: {self.title}")  
  
class EBook(Book):  
   def \_\_init\_\_(self, title, file\_size):  
       super().\_\_init\_\_(title)  
       self.file\_size = file\_size  
  
   def show(self):  
       super().show()  
       print(f"File Size: {self.file\_size} MB")  
  
eb = EBook("Python Guide", 5)  
eb.show()

🡪 Book stores the title and shows it using show().  
🡪 EBook is a type of Book that also stores file size in MB.  
🡪eb.show() prints the title first, then the file size.1. Book stores the title and shows it using show().  
o/p: Title : Python Guide

File Size: 5 MB