# **Name: Abdurrahman Qureshi**

# **Roll No: 242466**

Practical No: 11

**Q) To demonstrate following OOP concept in python**

1. **Class & Object.**
2. **Constructor.**
3. **Class variable / Class method.**
4. **Inheritance.**
5. **Super method.**

**Write the code and explain your code in your own words.**

a) Class & Object.

CODE:

class Specialist:

    def \_\_init\_\_(self, name, ability, weapon, quote):

        self.name = name

        self.ability = ability

        self.weapon = weapon

        self.quote = quote

    def show\_specialist(self):

        print(f"Specialist: {self.name}")

        print(f"Ability: {self.ability}")

        print(f"Weapon: {self.weapon}")

        print(f"Quote: '{self.quote}'")

spectre = Specialist("Spectre", "Active Camo", "Ripper", "I am the shadow, and the knife.")

seraph = Specialist("Seraph", "Annihilator", "Annihilator Pistol", "Justice is not blind, it is precise.")

outrider = Specialist("Outrider", "Vision Pulse", "Sparrow", "I see you, even in the dark.")

battery = Specialist("Battery", "Kinetic Armor", "War Machine", "You can't outrun the inevitable.")

spectre.show\_specialist()

print()

seraph.show\_specialist()

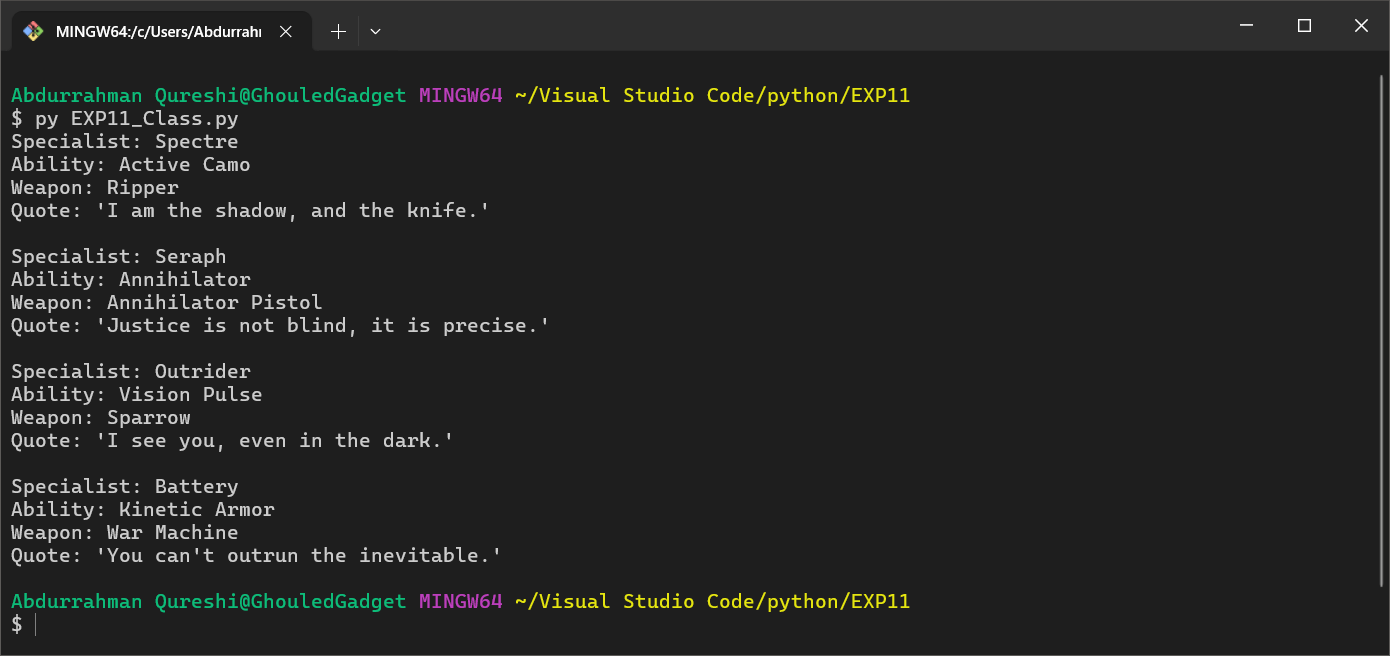
print()

outrider.show\_specialist()

print()

battery.show\_specialist()

OUTPUT:



EXPLANATION:

A class for BlackOps3 Specialists. Demonstrates **encapsulation** with attributes (name, ability, weapon, quote) and a method (show**\_**specialist). Instances (spectre, seraph) represent objects with unique abilities and quotes.

b) Constructor

CODE:

class ResidentEvilCharacter:

    def \_\_init\_\_(self, name, ability, weapon):

        self.name = name

        self.ability = ability

        self.weapon = weapon

    def show\_character(self):

        print(f"Character: {self.name}")

        print(f"Ability: {self.ability}")

        print(f"Weapon: {self.weapon}")

ethan = ResidentEvilCharacter("Ethan Winters", "Regeneration", "Lemi Handgun")

lady\_dimitrescu = ResidentEvilCharacter("Lady Dimitrescu", "Super Strength", "Claw Attacks")

heisenberg = ResidentEvilCharacter("Karl Heisenberg", "Magnetic Control", "Hammer")

ethan.show\_character()

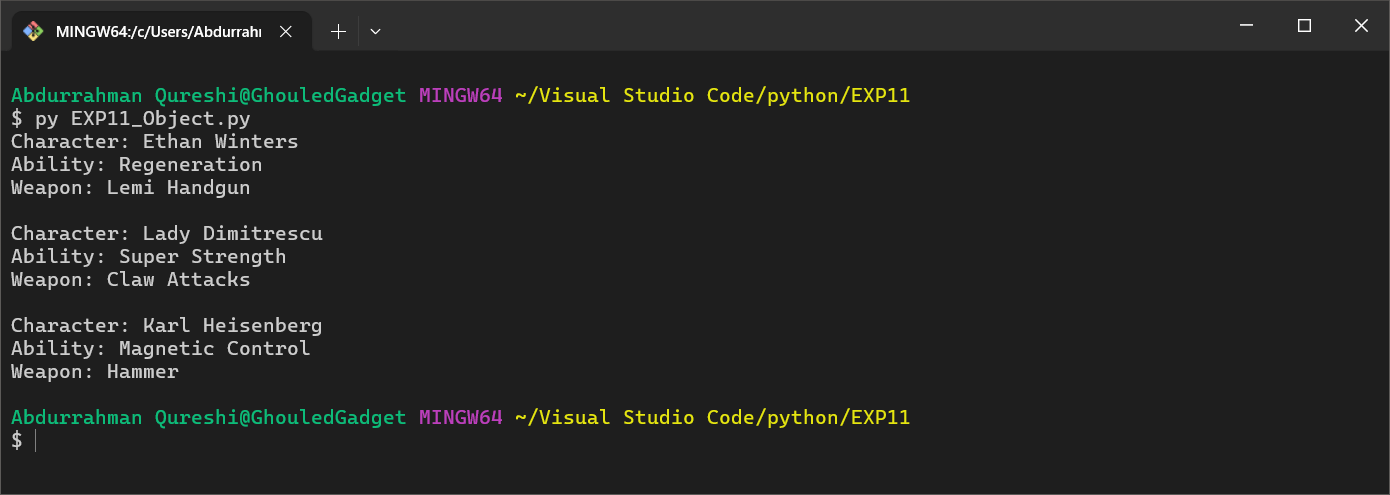
print()

lady\_dimitrescu.show\_character()

print()

heisenberg.show\_character()

OUTPUT:



EXPLANATION:

A class encapsulating Resident Evil characters. Demonstrates encapsulation with attributes (name, ability, weapon) and a method (show\_character). Instances (ethan, lady\_dimitrescu) represent objects with unique properties.

c) Class variable / Class method.

CODE:

class RDR2Character:

    game = "Red Dead Redemption II"

    characters = {

        "Arthur Morgan": {

            "gang": "Van der Linde Gang",

            "role": "Enforcer",

            "signature\_weapon": "Cattleman Revolver"

        },

        "Dutch van der Linde": {

            "gang": "Van der Linde Gang",

            "role": "Leader",

            "signature\_weapon": "Double-Action Revolver"

        },

        "John Marston": {

            "gang": "Van der Linde Gang",

            "role": "Gunslinger",

            "signature\_weapon": "Schofield Revolver"

        },

        "Sadie Adler": {

            "gang": "Van der Linde Gang",

            "role": "Bounty Hunter",

            "signature\_weapon": "Lancaster Repeater"

        }

    }

    @classmethod

    def print\_characters(cls):

        for character, details in cls.characters.items():

            print(f"{character}:")

            print(f"   Gang: {details["gang"]}")

            print(f"   Role: {details["role"]}")

            print(f"   Signature Weapon: {details["signature\_weapon"]}")

            print()

print()

print("------------------------------------------------------------")

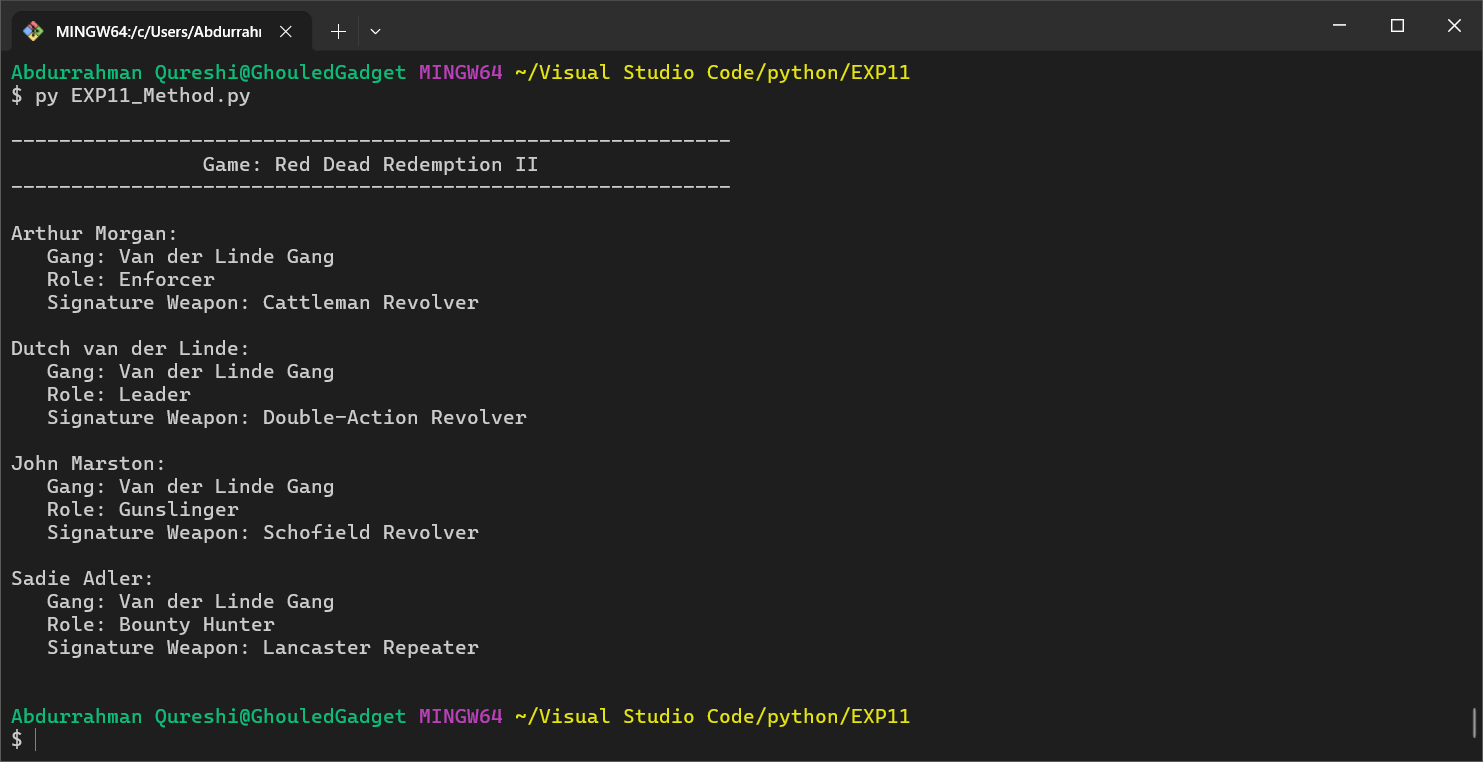
print("\t\tGame:" , RDR2Character.game)

print("------------------------------------------------------------")

print()

RDR2Character.print\_characters()

OUTPUT:



EXPLANATION:

A class for Red Dead Redemption 2 characters. Uses the concept of ‘class variables’ and ‘class methods’, that are ‘game’, ‘characters’and ‘print\_characters’. Without instantiating and object, we call the ‘print\_characters’class method.

d) Inheritance.

CODE:

class Character:

    def \_\_init\_\_(self, name, role, weapon):

        self.name = name

        self.role = role

        self.weapon = weapon

    def show\_info(self):

        print(f"Name: {self.name}")

        print(f"Role: {self.role}")

        print(f"Weapon: {self.weapon}")

class Human(Character):

    def \_\_init\_\_(self, name, role, weapon, tribe):

        super().\_\_init\_\_(name, role, weapon)

        self.tribe = tribe

    def show\_info(self):

        super().show\_info()

        print(f"Tribe: {self.tribe}")

class Machine(Character):

    def \_\_init\_\_(self, name, role, weapon, weakness):

        super().\_\_init\_\_(name, role, weapon)

        self.weakness = weakness

    def show\_info(self):

        super().show\_info()

        print(f"Weakness: {self.weakness}")

class Tribe(Character):

    def \_\_init\_\_(self, name, role, weapon, territory):

        super().\_\_init\_\_(name, role, weapon)

        self.territory = territory

    def show\_info(self):

        super().show\_info()

        print(f"Territory: {self.territory}")

aloy = Human("Aloy", "Hunter", "Bow", "Nora")

thunderjaw = Machine("Thunderjaw", "Apex Predator", "Disc Launcher", "Tearblast Arrows")

tenakth = Tribe("Tenakth", "Warrior Tribe", "Spear", "The Clan Lands")

print("--- Human ---")

aloy.show\_info()

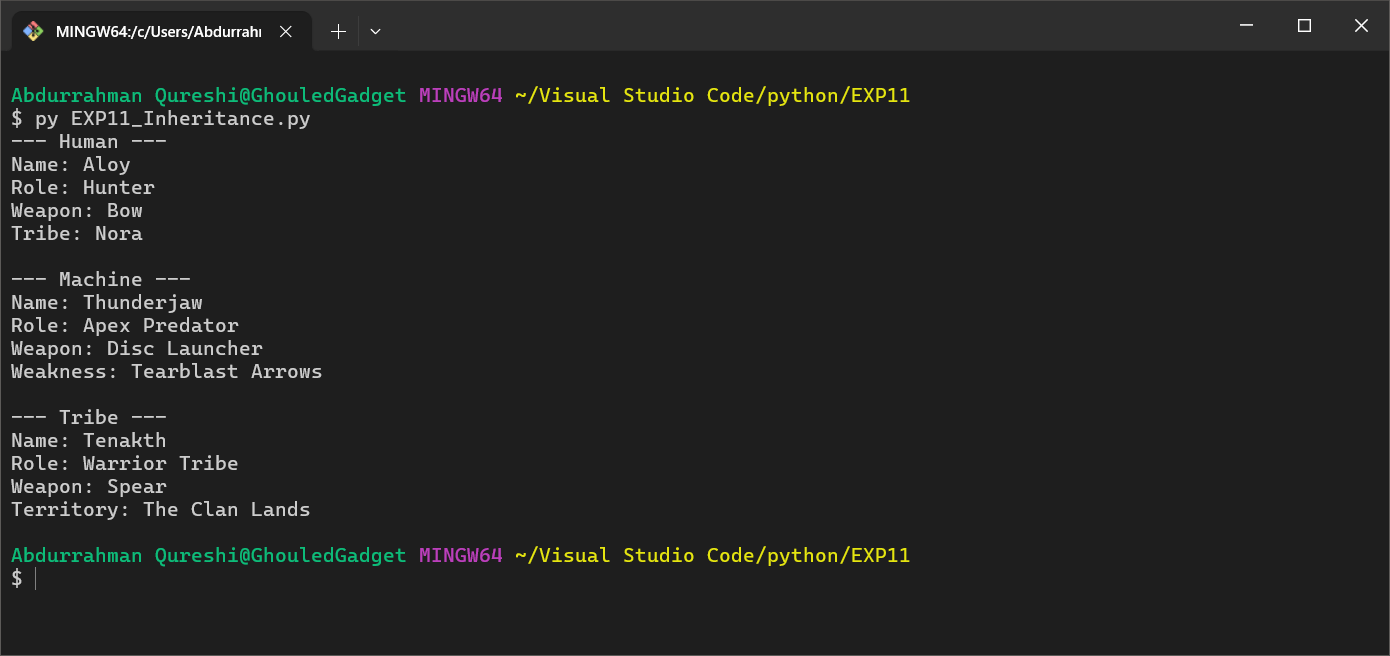
print("\n--- Machine ---")

thunderjaw.show\_info()

print("\n--- Tribe ---")

tenakth.show\_info()

OUTPUT:



EXPLANATION:

A base class for characters. Demonstrates inheritance with subclasses (Human, Machine, Tribe). Each subclass extends the base class with additional attributes (tribe, weakness, territory) and overrides show\_info for polymorphism.

e) Super Keyword.

CODE:

class Soldier:

    def \_\_init\_\_(self, name, health, weapon):

        self.name = name

        self.health = health

        self.weapon = weapon

    def show\_info(self):

        print(f"Name: {self.name}")

        print(f"Health: {self.health}")

        print(f"Weapon: {self.weapon}")

    def attack(self):

        print(f"{self.name} is attacking with {self.weapon}!")

class Assault(Soldier):

    def \_\_init\_\_(self, name, health, weapon, ability):

        super().\_\_init\_\_(name, health, weapon)

        self.ability = ability

    def show\_info(self):

        super().show\_info()

        print(f"Ability: {self.ability}")

    def use\_ability(self):

        print(f"{self.name} is using {self.ability}!")

class Sniper(Soldier):

    def \_\_init\_\_(self, name, health, weapon, scope):

        super().\_\_init\_\_(name, health, weapon)

        self.scope = scope

    def show\_info(self):

        super().show\_info()

        print(f"Scope: {self.scope}x")

    def snipe(self):

        print(f"{self.name} is sniping with a {self.scope}x scope!")

assault\_soldier = Assault("John", 100, "M4A1", "Tactical Sprint")

sniper\_soldier = Sniper("Alex", 80, "AX-50", 10)

print("--- Assault Soldier ---")

assault\_soldier.show\_info()

assault\_soldier.use\_ability()

assault\_soldier.attack()

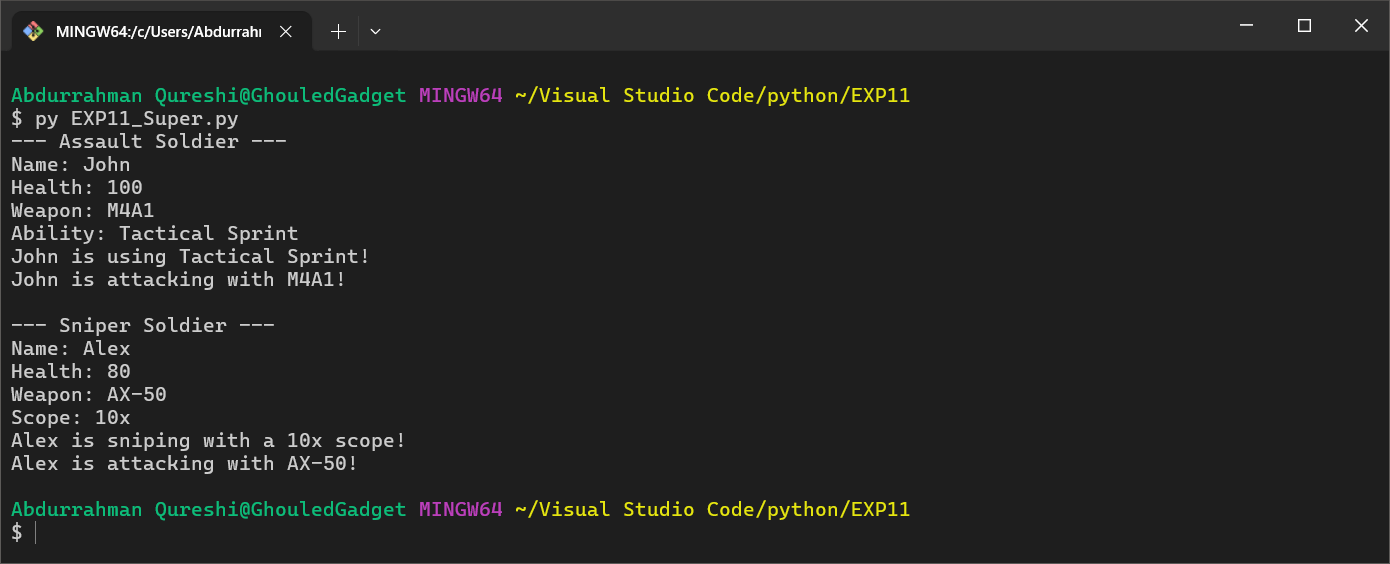
print("\n--- Sniper Soldier ---")

sniper\_soldier.show\_info()

sniper\_soldier.snipe()

sniper\_soldier.attack()

OUTPUT:



EXPLANATION:

A base class for soldiers. Demonstrates inheritance with subclasses (Assault, Sniper). Uses super() to call parent methods, showcasing method overriding and polymorphism. Instances (assault\_soldier, sniper\_soldier) represent specialized soldiers.