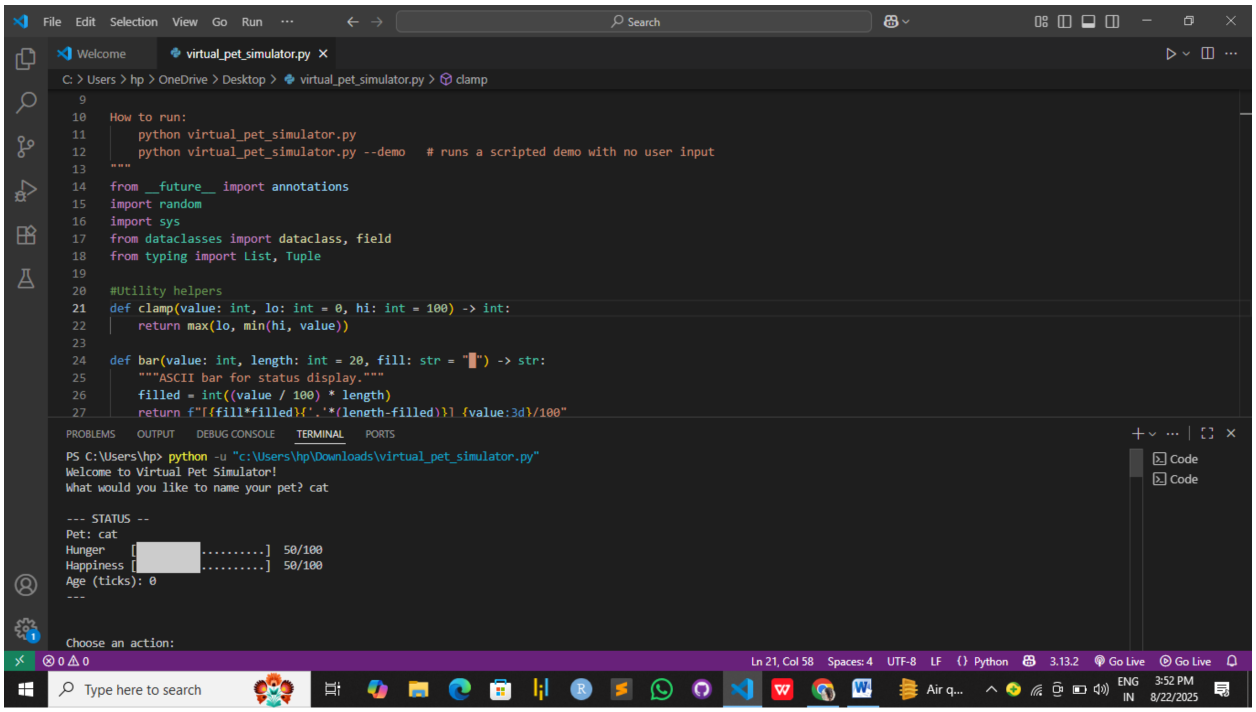
Assignment. 4:-

Virtual Pet Simulator

Code :-

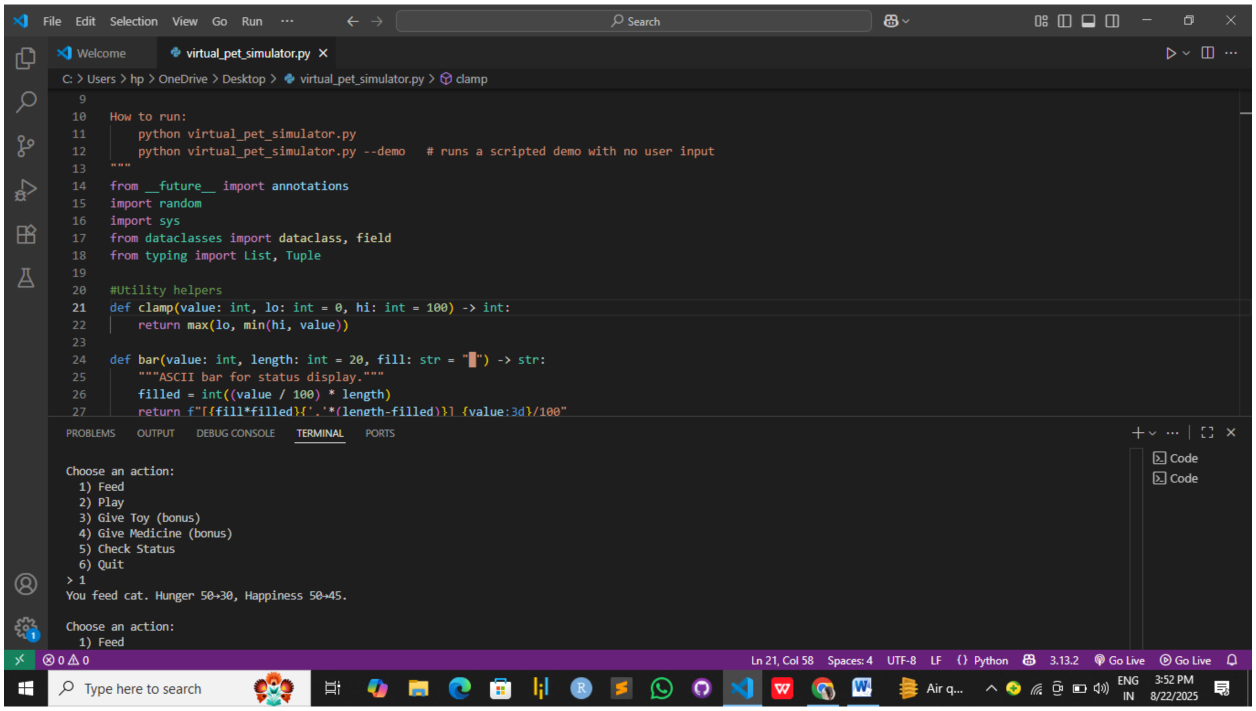
```python
"""
Virtual Pet Simulator
- Command-line game where you feed and play with a virtual pet.
- Tracks hunger and happiness (0..100).
- Auto "tick" lowers happiness & raises hunger over time.
- Game over if hunger hits 100 or happiness hits 0.
- Bonus: name your pet, random events, toy/medicine actions, and demo mode.

How to run:
    python virtual_pet_simulator.py
    python virtual_pet_simulator.py --demo   # runs a scripted demo with no user
input
"""
from __future__ import annotations
import random
import sys
from dataclasses import dataclass, field
from typing import List, Tuple

#Utility helpers
def clamp(value: int, lo: int = 0, hi: int = 100) -> int:
    return max(lo, min(hi, value))

def bar(value: int, length: int = 20, fill: str = "█") -> str:
    """ASCII bar for status display."""
    filled = int((value / 100) * length)
    return f"[{fill*filled}{'.'*(length-filled)}] {value:3d}/100"


@dataclass
class VirtualPet:
    name: str
    hunger: int = 50
    happiness: int = 50
    age_ticks: int = 0
     # how many ticks have passed (for auto changes pacing)
    action_count: int = 0
     # user actions since last auto change
    event_log: List[str] = field(default_factory=list)

    def log(self, message: str) -> None:
        self.event_log.append(message)
        print(message)

    #Core mechanics
    def feed(self) -> None:
        """Feeding decreases hunger, but slightly decreases happiness (pet may be
sleepy)."""
        old_hunger, old_happiness = self.hunger, self.happiness
        self.hunger = clamp(self.hunger - 20)
        self.happiness = clamp(self.happiness - 5)
        self.log(f"You feed {self.name}. Hunger {old_hunger}→{self.hunger},
Happiness {old_happiness}→{self.happiness}.")
        self.after_action()

    def play(self) -> None:
        """Playing increases happiness, but slightly increases hunger."""
        old_hunger, old_happiness = self.hunger, self.happiness
        self.happiness = clamp(self.happiness + 20)
        self.hunger = clamp(self.hunger + 10)
        self.log(f"You play with {self.name}! Happiness
{old_happiness}→{self.happiness}, Hunger {old_hunger}→{self.hunger}.")
        self.after_action()

    def give_toy(self) -> None:
        """Bonus action: toy gives a small happiness boost with no hunger change
(limited effect)."""
        old = self.happiness
        self.happiness = clamp(self.happiness + 10)
        self.log(f"You give {self.name} a toy. Happiness
{old}→{self.happiness}.")
        self.after_action()

    def give_medicine(self) -> None:
        """Bonus action: medicine calms hunger spikes but slightly reduces
happiness."""
        old_hunger, old_happiness = self.hunger, self.happiness
        self.hunger = clamp(self.hunger - 10)
        self.happiness = clamp(self.happiness - 3)
        self.log(f"You give medicine. Hunger {old_hunger}→{self.hunger},
Happiness {old_happiness}→{self.happiness}.")
        self.after_action()
    def status(self) -> None:
        """Display current status with bars and notes."""
        print("\n--- STATUS --")
        print(f"Pet: {self.name}")
        print(f"Hunger    {bar(self.hunger)}")
        print(f"Happiness {bar(self.happiness)}")
        print(f"Age (ticks): {self.age_ticks}")
        if self.hunger > 80:
            print(f"Warning: {self.name} is very hungry! Happiness will drop.")
        if self.happiness < 20:
            print(f"Uh oh: {self.name} is getting sad. Time to play!")
        print("---\n")

    def after_action(self) -> None:
        """Apply post-action rules and pacing."""
        self.action_count += 1
        # If hunger is too high, happiness decreases (requirement)
        if self.hunger > 80:
            old = self.happiness
            self.happiness = clamp(self.happiness - 10)
            self.log(f"{self.name} feels grumpy from hunger. Happiness
{old}→{self.happiness}.")
        # Periodic auto changes every 2 actions
        if self.action_count % 2 == 0:
            self.tick()

    def tick(self) -> None:
        """
        Automatic time passage:
          hunger up, happiness down.
          """
        self.age_ticks += 1
        old_hunger, old_happiness = self.hunger, self.happiness
        self.hunger = clamp(self.hunger + 5)
        self.happiness = clamp(self.happiness - 3)
        self.log(f"Time passes... Hunger {old_hunger}→{self.hunger}, Happiness
{old_happiness}→{self.happiness}.")
        self.random_event()

    # ---- Random events (bonus) ----
    def random_event(self) -> None:
        roll = random.random()
        if roll < 0.12:
            # Finds a snack
            old = self.hunger
            self.hunger = clamp(self.hunger - 8)
            self.log(f"Lucky! {self.name} found a snack. Hunger
{old}→{self.hunger}.")
        elif roll < 0.20:
            # Mini-zoomies
            old_happiness = self.happiness
            self.happiness = clamp(self.happiness + 6)
            self.log(f"{self.name} has the zoomies! Happiness
{old_happiness}→{self.happiness}.")
        elif roll < 0.24:
            # Mild sickness
            old = self.happiness
            self.happiness = clamp(self.happiness - 6)
            self.log(f"Oh no, {self.name} feels a bit under the weather.
Happiness {old}→{self.happiness}.")

    #End conditions
    def is_game_over(self) -> Tuple[bool, str]:
        if self.hunger >= 100:
            return True, f"{self.name} became too hungry. Game over."
        if self.happiness <= 0:
            return True, f"{self.name} became too sad. Game over."
        return False, ""


#Game loop
MENU = """
Choose an action:
  1) Feed
  2) Play
  3) Give Toy (bonus)
  4) Give Medicine (bonus)
  5) Check Status
  6) Quit
> """

def run_game() -> None:
    print("Welcome to Virtual Pet Simulator!")
    name = input("What would you like to name your pet? ").strip() or "Buddy"
    pet = VirtualPet(name=name)

    pet.status()
    while True:
        choice = input(MENU).strip()
        if choice == "1":
            pet.feed()
        elif choice == "2":
            pet.play()
        elif choice == "3":
            pet.give_toy()
        elif choice == "4":
            pet.give_medicine()
        elif choice == "5":
            pet.status()
        elif choice == "6":
            print("Thanks for playing! Bye!")
            break
        else:
            print("Please choose a valid option (1-6).")
            continue

        over, msg = pet.is_game_over()
        if over:
            print(msg)
            break

#Demo (non-interactive)
def run_demo() -> None:
    random.seed(7)  # deterministic demo
    pet = VirtualPet(name="Pixel", hunger=50, happiness=50)
    transcript = []
    def capture(msg: str):
        transcript.append(msg)

    # Monkey-patch log to capture and print
    original_log = pet.log
    def log_and_capture(message: str) -> None:
        capture(message)
        original_log(message)
    pet.log = log_and_capture  # type: ignore

    # Scripted sequence of actions
    actions = [
        ("status", None),
        ("play", None),
        ("feed", None),
        ("play", None),
        ("give_toy", None),
        ("status", None),
        ("feed", None),
        ("give_medicine", None),
        ("play", None),
        ("status", None),
    ]
    print("\n--- DEMO RUN START ---\n")
    for action, arg in actions:
        if action == "play":
            pet.play()
        elif action == "feed":
            pet.feed()
        elif action == "give_toy":
            pet.give_toy()
        elif action == "give_medicine":
            pet.give_medicine()
        elif action == "status":
            pet.status()
        over, msg = pet.is_game_over()
        if over:
            print(msg)
            break
    print("\n--- DEMO RUN END ---\n")

    # Save transcript to file for assignment evidence
    path = "/mnt/data/sample_run.txt"
    with open(path, "w", encoding="utf-8") as f:
        f.write("Virtual Pet Simulator — Sample Demo Transcript\n")
        f.write("==\n\n")
        for line in transcript:
            f.write(line + "\n")
    print(f"Saved demo transcript to: {path}")

if __name__ == "__main__":
    if "--demo" in sys.argv:
        run_demo()
    else:
        run_game()
```

Result :-



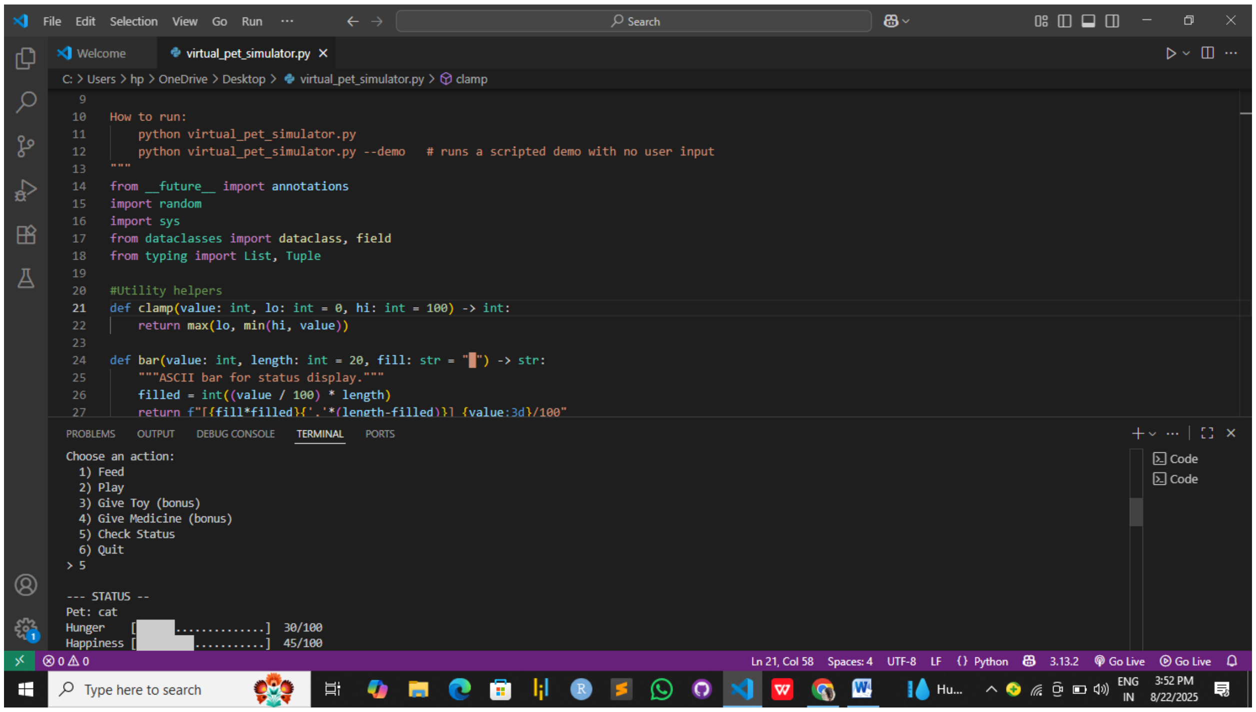
```python

How to run:
    python virtual_pet_simulator.py
    python virtual_pet_simulator.py --demo   # runs a scripted demo with no user input
"""
from __future__ import annotations
import random
import sys
from dataclasses import dataclass, field
from typing import List, Tuple

#Utility helpers
def clamp(value: int, lo: int = 0, hi: int = 100) -> int:
    return max(lo, min(hi, value))

def bar(value: int, length: int = 20, fill: str = "█") -> str:
    """ASCII bar for status display."""
    filled = int((value / 100) * length)
    return f"[{fill*filled}{'.'*(length-filled)}] {value:3d}/100"
```

```
PS C:\Users\hp> python -u "c:\Users\hp\Downloads\virtual_pet_simulator.py"
Welcome to Virtual Pet Simulator!
What would you like to name your pet? cat

--- STATUS --
Pet: cat
Hunger    [██████████..........]  50/100
Happiness [██████████..........]  50/100
Age (ticks): 0
---


Choose an action:
```



```python

How to run:
    python virtual_pet_simulator.py
    python virtual_pet_simulator.py --demo   # runs a scripted demo with no user input
"""
from __future__ import annotations
import random
import sys
from dataclasses import dataclass, field
from typing import List, Tuple

#Utility helpers
def clamp(value: int, lo: int = 0, hi: int = 100) -> int:
    return max(lo, min(hi, value))

def bar(value: int, length: int = 20, fill: str = "█") -> str:
    """ASCII bar for status display."""
    filled = int((value / 100) * length)
    return f"[{fill*filled}{'.'*(length-filled)}] {value:3d}/100"
```

```

Choose an action:
  1) Feed
  2) Play
  3) Give Toy (bonus)
  4) Give Medicine (bonus)
  5) Check Status
  6) Quit
> 1
You feed cat. Hunger 50→30, Happiness 50→45.

Choose an action:
  1) Feed
```



```python

How to run:
    python virtual_pet_simulator.py
    python virtual_pet_simulator.py --demo   # runs a scripted demo with no user input
"""
from __future__ import annotations
import random
import sys
from dataclasses import dataclass, field
from typing import List, Tuple

#Utility helpers
def clamp(value: int, lo: int = 0, hi: int = 100) -> int:
    return max(lo, min(hi, value))

def bar(value: int, length: int = 20, fill: str = "█") -> str:
    """ASCII bar for status display."""
    filled = int((value / 100) * length)
    return f"[{fill*filled}{'.'*(length-filled)}] {value:3d}/100"
```

```
Choose an action:
  1) Feed
  2) Play
  3) Give Toy (bonus)
  4) Give Medicine (bonus)
  5) Check Status
  6) Quit
> 5

--- STATUS --
Pet: cat
Hunger    [██████..............]  30/100
Happiness [█████████...........]  45/100
```

GitHub Deployment Link :-

https://github.com/Neha-yadav-Full/VaultOfCodes-Internship