PRACTICE 3: Lab3 Drunkers

Date: *27 of June of 2025*

Professor: Cesar Eduardo Inda Cisneros

Students:

- JHONAT FERNANDO ACEVEDO POLANCO
- ALEJANDRO REAL LOPEZ
- LUIS ANTONIO CASILLAS DE LA CRUZ
- SCARLET ELIZABETH LAMAS VILLALOBOS
- JONATHAN EDUARDO ORTEGA BAÑUELOS

Team: Oreo

INTRODUCTION

In this practice session, we created a program that recreates the behavior of a group of drunkers at a party, where each person performs actions typical of a fun and wild night. The code, developed in Python, uses data structures such as dictionaries and lists to record each character's past actions, ensuring that certain key activities are not repeated in consecutive cycles.

In addition, functions are implemented that represent specific behaviors (such as drinking beer, using the bathroom, or calling an ex).

This project not only serves as a practical example of structured programming in Python, but also as a demonstration of how programming logic can be applied to model everyday situations in a dynamic and controlled manner.

REQUERIMENTS

1.- Add 5 "Drunkers"

2.- Add 4 "Actions"

3.- 4 Cycles execution , Each cycle must be different.

DEVELOPMENT

These lines bring in built-in Python modules. Time lets us add delays like pausing a program to make it more realistic. Random is used when we want things to happen randomly, like picking someone at random to do something.

```
import time
import random
```

This part sets up the main party crew just a list of names called borrachos. Then there's a dictionary called acciones_pasadas that tracks what each guy has done so far (like going to the bathroom or calling their ex). That way, they don't repeat the same things in later rounds.

```python
# Lista de nombres de los borrachos
borrachos = ["Cesar", "Victor", "Alejandro", "Andres", "Jonathan"]

# Registro de acciones pasadas
"""Se realizo el map para que no repitan la misma accion en los diferentes ciclos.
De esta forma se tiene un log de las acciones pasadas de cada uno de los borrachitos."""
acciones_pasadas = {nombre: [] for nombre in borrachos} # Map con las acciones pasadas
```

This function is called tomar. It just prints that the person is drinking beer, then pauses for one second (to make it feel real-time), and then says they finished the beer.

```python
# Acciones disponibles
def tomar(nombre):
    print(f"{nombre} está tomando cerveza...")
    time.sleep(1)
    print(f"{nombre} se acabó la cerveza.")
```

This is the usar_baño function basically, it simulates when someone goes to the bathroom. It prints that the person is peeing, waits one second (for dramatic effect), and then prints that they're done.

```python
def usar_baño(nombre):
    print(f"{nombre} está orinando....")
    time.sleep(1)
    print(f"{nombre} salió del baño.")
```

This function is called call_ex, which is like the most chaotic part of a drunk night. It makes the dude (whoever's name is passed as nombre) call their ex while crying, then waits a second and prints that their ex hung up on them.

```python
def call_ex(nombre):
    print(f"{nombre} está llamando a su ex llorando...")
    time.sleep(1)
    print(f"{nombre} le colgó su ex y sigue llorando.")
```

This function's called singing, and it's all about when one of the borrachos suddenly breaks in to sing. It prints a message saying the person's singing, waits one second to let that moment land, and that's it.

```python
def singing(nombre):
    print(f"{nombre} está cantando a todo pulmón!")
    time.sleep(1)
```

This part loops through each person in the party (for borracho in borrachos:) and assigns actions:

- If the person is the one chosen to use the bathroom (bañero), it calls the usar_baño() function.
- If it's the one calling their ex (llamador), it calls call_ex().
- Everyone else gets a random action: either tomar() (drink) or singing() (sing), selected using random.choice().

It finishes with a short delay between each person's turn to make the simulation feel more natural and spaced out.

```
32  def main():
33      print("¡Inicia la fiesta de los borrachos!\n")
```

This code runs the main simulation loop four times (using range(4)), where each round is a new party cycle. At the start of each cycle, it:

- Prints which cycle is starting.
- Creates a list of people (candidatos_baño) who haven't yet used the bathroom, using a list comprehension.
- Randomly picks one of them (bañero) and logs that they used the bathroom by updating their entry in acciones_pasadas.

This setup helps keep the simulation dynamic and ensures no one repeats key actions too early

```
35      for ciclo in range(4):
36          print(f"\n=== CICLO {ciclo + 1} ===")
37          """En los list comprehension de candidatos se realiza para cada borrachito
38          en la lista de borrachos tomando como base el diccionario de acciones pasadas
39          functiona de tal forma que checa si el borrachito no tiene baño o llamar en acciones pasadas."""
40          candidatos_baño = [b for b in borrachos if 'baño' not in acciones_pasadas[b]] # List comprehension
41          bañero = random.choice(candidatos_baño) # De forma aleatoria se toma un candidato de baño
42          acciones_pasadas[bañero].append('baño') # Se guarda la accion de llamar en el nombre del borrachito
```

This section filters out which "borrachos" (party guests) haven't called their ex yet and aren't the person who just went to the bathroom (bañero). It:

Uses a list comprehension to build candidatos_llamar—only those who haven't done the 'llamar' action and aren't the bañero.

Picks one randomly with random.choice() and saves that they called their ex by appending 'llamar' to their record in acciones_pasadas.

```
        candidatos_llamar = [b for b in borrachos if b != bañero and 'llamar' not in acciones_pasadas[b]] # List Comprehension
        llamador = random.choice(candidatos_llamar) # De forma aleatoria se toma un candidato de llamar
        acciones_pasadas[llamador].append('llamar') # Se guardan la accion de llamar en el nombre del borrachito
```

This block executes each person's turn in the party:

- Iterates through all members in borrachos.
- If someone is the selected bañero, they run usar_baño().
- If they're the llamador, they run call_ex().
- Otherwise, they randomly do either tomar() or singing()—these don't get logged since they're repeatable fun actions.
- Adds a 2-second pause between each action with time.sleep(2) to make everything feel spaced out.

At the end, it prints a message wrapping up the current cycle, waits a second, then prints a final line signaling that the whole event is over.

```
        for borracho in borrachos:
            if borracho == bañero:
                usar_baño(borracho)
            elif borracho == llamador:
                call_ex(borracho)
            else:
                # Acciones alternativas (no se registran porque no son únicas)
                random.choice([tomar, singing])(borracho)

            time.sleep(2)  # Pausa entre los borrachos

        print(f"\n--- Fin del ciclo {ciclo + 1} ---")
        time.sleep(1)

    print("\n¡La fiesta ha terminado! Todos los borrachos se fueron a casa.")
```

if __name__ == "__main__": ensures the following code only runs when the file is executed directly, not when imported elsewhere. main() gets called to kick off the whole party simulation.

Basically, it's the "entry point" that launches everything.

```
if __name__ == "__main__":
    main()
```

RESULTS

The program successfully ran a party simulation with five drunk characters going through four party cycles. Each round, one person went to the bathroom, another called their ex, and the rest either drank or sang. The output logs every action, cycle by cycle, with perfect comedic timing. It ends with a fun message: "`¡La fiesta ha terminado! Todos los borrachos se fueron a casa`."

```
=== CICLO 1 ===
Cesar está orinando....
Cesar salió del baño.
Victor está llamando a su ex llorando...
Victor le colgó su ex y sigue llorando.
Alejandro está tomando cerveza...
Alejandro se acabó la cerveza.
Andres está tomando cerveza...
Andres se acabó la cerveza.
Jonathan está tomando cerveza...
Jonathan se acabó la cerveza.

--- Fin del ciclo 1 ---

=== CICLO 2 ===
Cesar está cantando a todo pulmón!
Victor está orinando....
Victor salió del baño.
Alejandro está cantando a todo pulmón!
Andres está cantando a todo pulmón!
Jonathan está llamando a su ex llorando...
Jonathan le colgó su ex y sigue llorando.

--- Fin del ciclo 2 ---

=== CICLO 3 ===
Cesar está llamando a su ex llorando...
Cesar le colgó su ex y sigue llorando.
Victor está cantando a todo pulmón!
Alejandro está orinando....
Alejandro salió del baño.
Andres está cantando a todo pulmón!
Jonathan está tomando cerveza...
Jonathan se acabó la cerveza.

--- Fin del ciclo 3 ---

=== CICLO 4 ===
Cesar está tomando cerveza...
Cesar se acabó la cerveza.
Victor está tomando cerveza...
Victor se acabó la cerveza.
Alejandro está tomando cerveza...
Alejandro se acabó la cerveza.
Andres está llamando a su ex llorando...
Andres le colgó su ex y sigue llorando.
Jonathan está orinando....
Jonathan salió del baño.

--- Fin del ciclo 4 ---

¡La fiesta ha terminado! Todos los borrachos se fueron a casa.
```

ISSUES

In this practice we have some troubles in how to random the drunkers without repeat the same drunker in another cycle, that's why we decided to incorporate a map or how it is called in python a dictionary, called the acciones_pasadas, where it will be the drunkers and also all the activities done by them. With this we solve the part to give in each cycle and each drunker a different activity to do.

In the code we can see many lists done in list comprehension. We did it in this way to reduce the code programmed, but on the other hand it could be more difficult to understand what is going on in the list comprehension, that's why we add the comments in each list.

In summary, those were the issues we have coding and thinking about this program.

CONCLUSIONS

The biggest challenge of this project was controlling actions so that no one would repeat specific ones, like going to the bathroom or calling their ex.

We solved this by using a dictionary that keeps a history of what each person has already done. So, in each cycle, the program first filtered out those who hadn't yet performed those actions and then randomly chose someone from only the available candidates.

-Ortega Bañuelos Jonathan Eduardo.

The logic behind how concurrency works, or how to create pseudo-concurrency, was a challenge. Thinking about how we should implement the different parts, putting them together, and ensuring optimal code synergy was difficult.

I feel like we overcomplicated certain parts of the code, but I feel that in the end, concurrency was achieved, giving us a better understanding of how different systems work internally.

-Acevedo Polanco Jhonat Fernando.

The most interesting part of the code was using random to completely randomize who performed what action. The goal was to create a code that was functional and short on lines. Another challenge was ensuring that actions weren't repeated more than once. However, it was very interesting to see how certain issues were resolved.

-Lamas Villalobos Scarlet Elizabeth.

In conclusion, developing this code was a super fun and personally rewarding experience. It started as a challenge from my teacher to simulate a party with some classic drunk behavior, but it evolved into an ingenious system with memory, randomness, and structured logic. I really enjoyed it because I was able to practice using dictionaries, functions, list comprehensions, and randomization in a fun and meaningful way. In short, it was an excellent way to apply Python to something creative and lighthearted. I learned a lot while having fun, although there were some challenges, but it was definitely a success.

-Alejandro Real Lopez.

It was a very hard challenge but I like it to.

-Luis Antonio Casillas De La Cruz.

GITHUB REPOSITORY

The url for see the project is: https://github.com/ISOFT5-SA/ISOFT5_SA/tree/LAB-3