Lab 3: KingKizz

Nombre: Ever Hemerson Ceballos Rivera

Entregado: Julián Darío Barrero, Alexander López Parrado

Universidad del Quindío

**resumen.**

En este laboratorio aborda el desarrollo de un servidor para gestionar jugadores y preguntas, utilizando archivos y estructuras de programación en Python, para esto se suministró unos códigos base que no debe ser modificado **trivia\_server.py, trivia\_client.py, test\_trivia\_client.py,** y el objetivo es implementar funciones para registrar usuarios, actualizar los puntajes, verificar usuarios en línea y gestión de preguntas. Se evaluará la funcionalidad del servidor en un entorno de red local o en el mismo equipo, utilizando pruebas específicas implementando las funciones del código **users.py.**

***Palabras clave:*** *Programación Orientada a Objetos, Python, Clases, Métodos, Excepciones, GUI, Trivia, Gestión de usuarios.*

**introducción.**

La programación orientada a objetos (POO) es un paradigma de diseño que organiza el código en torno a **objetos.** Para esto se ha venido usando Python, como lenguaje de uso general, ofrece una mejor implementación para la POO, el cuál facilita el desarrollo de aplicaciones modulares. En este informe se presentan dos programas orientados a objetos que emplean bibliotecas como **tkinter** y **Pillow**, además de módulos personalizados como las funciones de **Trivia\_client**, la interfaz y la ruleta para el usuario y **users.py**, las preguntas.txt y **Trivia\_server** para la comunicación entre el usuario y el servidor.

**Requisitos técnicos.**

* **Lenguaje de programación:** Python, con enfoque en programación orientada a objetos (POO).
* **Estructura del proyecto:** Modular, con uso de clases y métodos que implementen conceptosde POO como herencia, encapsulamiento y polimorfismo.
* **Archivos externos:** Uso de archivos.txt para almacenar las preguntas y categorías del juego.
* **Interfaz:** Implementación de una ruleta para la selección de categorías y gestión de preguntas.
* **Gestión de errores:** Manejo de excepciones para asegurar la robustez del programa.
* **Reutilización del código:** Diseño que permita la extensión o modificación sin afectar lafuncionalidad existente como **users.py** del anterior laboratorio y **Trivia\_client** para que estas funciones interactúen con las funciones del servidor.

**Entrega.**

El informe y el código implementado se deben entregar a través de un repositorio en GitHub y compartirse en la plataforma Google Classroom.

1. **OBJETIVOS.**

**Objetivo General.**

Crear una interfaz gráfica interactiva para jugar trivia que incluya funcionalidades de registro de usuarios, inicio de sesión, obtención de puntaje, actualización de puntaje, cierre de sesión, selección de categorías y preguntas aleatorias, adaptada para conectarse con varios computadores.

**Objetivos Específicos.**

* Permitir el registro y gestión de usuarios mediante un archivo local de texto **(usuarios.txt).**
* Implementar una ruleta gráfica que seleccione categorías de forma aleatoria.
* Mostrar preguntas específicas según la categoría seleccionada y permitir la evaluación de las respuestas del jugador.
* Registrar y actualizar el puntaje de los jugadores en un archivo local.
* Cerrar correctamente la sesión y mediante un archivo local **(usuarios.txt)** aparezca desconectado el jugador
* Desplegar un sistema visual atractivo y funcional, utilizando la biblioteca Tkinter y soporte gráfico adicional.

1. **procedimiento y resultados.**

**PROCEDIMIENTOS**

**PROCEDIMIENTO DE LA RULETA**

El archivo ruleta.py define la clase Ruleta, que implementa una interfaz gráfica para el juego mediante una ruleta animada en Tkinter. Al inicializarse, configura una ventana con un lienzo para la ruleta, botones interactivos y variables como velocidad **(self.speed)** y ángulo **(self.angle).** El giro se inicia con el método **iniciar\_ruleta**, que activa la animación en **animar\_flecha,** donde la ruleta rota progresivamente más lento hasta detenerse. Una vez detenida, **finalizar\_ruleta** determina la categoría seleccionada según el ángulo final, actualiza la interfaz y carga una pregunta de esa categoría usando **mostrar\_pregunta.** Las preguntas se leen de un archivo local específico por categoría y se muestran en una ventana secundaria con opciones interactivas. Según la respuesta del jugador, **elegir\_respuesta** verifica si es correcta, actualiza el puntaje en el archivo local **(usuarios.txt)** e informa al usuario con el botón de ver puntaje.

**CÓDIGO USER.PY**

import tkinter as tk

from PIL import Image, ImageTk

import math

from tkinter import messagebox

from trivia\_client import closeSession, getScore, updateScore

import random

import json

# Constantes para la ubicación de preguntas y servidor

Preguntas\_File = {

    "preguntas0.txt",  # Deportes

    "preguntas1.txt",  # Música

    "preguntas2.txt",  # Cultura General

    "preguntas3.txt",  # Cine y Películas

    "preguntas4.txt",  # Historia

    "preguntas5.txt"   # Videojuegos

}

Base\_Url = "http://localhost:80"

class Ruleta:

    def \_\_init\_\_(self, root, username, password, base\_url):

        self.root = root

        self.username = username

        self.password = password

        self.base\_url = base\_url

        self.puntaje = int(self.obtener\_puntaje())  # Obtener el puntaje inicial desde el servidor

        self.root.title("Ruleta de Trivia")

        self.root.geometry("700x800")

        self.frame\_ruleta = tk.Frame(self.root, bg="#f0f0f0")

        self.frame\_ruleta.pack(fill=tk.BOTH, expand=True)

        self.ruleta\_image = Image.open("Ruleta.png")

        self.ruleta\_image = self.ruleta\_image.resize((400, 400))

        self.ruleta\_photo = ImageTk.PhotoImage(self.ruleta\_image)

        self.canvas = tk.Canvas(self.frame\_ruleta, width=500, height=500, bg="#ffffff", bd=0, highlightthickness=0)

        self.canvas.pack(pady=20)

        self.canvas.create\_image(250, 250, image=self.ruleta\_photo)

        self.angle = 0

        self.center\_x = 250

        self.center\_y = 250

        self.radius = 200

        self.flecha\_radius = 100

        self.initial\_speed = 30

        self.speed = 0

        self.deceleration = 0.5

        self.flecha = None

        self.girando = False

        self.categorias = ["Musica", "Deportes", "Cultura General",

                           "Cine y Peliculas", "Historia", "Videojuegos"]

        self.label\_categoria = tk.Label(self.frame\_ruleta, text="Selecciona una categoria", font=("Arial", 14), bg="#f0f0f0", fg="#333")

        self.label\_categoria.pack()

        self.boton\_iniciar = tk.Button(self.frame\_ruleta, text="Iniciar Ruleta", font=("Arial", 12), command=self.iniciar\_ruleta, bg="#4CAF50", fg="#fff")

        self.boton\_iniciar.pack(pady=10)

        self.boton\_ver\_puntaje = tk.Button(self.frame\_ruleta, text="Ver Puntaje", font=("Arial", 12), command=self.ver\_puntaje, bg="#008CBA", fg="#fff")

        self.boton\_ver\_puntaje.pack(pady=10)

        self.boton\_cerrar\_sesion = tk.Button(self.frame\_ruleta, text="Cerrar Sesion", font=("Arial", 12), command=self.cerrar\_sesion, bg="#f44336", fg="#fff")

        self.boton\_cerrar\_sesion.pack(pady=10)

    def obtener\_puntaje(self):

        """Obtiene el puntaje actual desde el servidor usando `getScore`."""

        try:

            puntaje = getScore(self.base\_url, self.username, self.password)

            return puntaje

        except Exception as e:

            print(f"Error al obtener el puntaje: {e}")

            return 0

    def actualizar\_puntaje(self, puntos):

        """Actualiza el puntaje en el servidor usando `updateScore`."""

        try:

            nuevo\_puntaje = self.puntaje + puntos

            print(f"Actualizando puntaje. Puntaje actual: {self.puntaje}, Nuevo puntaje: {nuevo\_puntaje}")

            mensaje = updateScore(self.base\_url, self.username, self.password, nuevo\_puntaje)

            print(f"Respuesta del servidor: {mensaje}")

            if isinstance(mensaje, str) and "Actualizado" in mensaje:

                self.puntaje = nuevo\_puntaje

                messagebox.showinfo("Puntaje Actualizado", "Tu puntaje ha sido actualizado a: " + str(self.puntaje))

            else:

                messagebox.showerror("Error", "No se pudo actualizar el puntaje. Respuesta: " + str(mensaje))

        except Exception as e:

            print(f"Error al actualizar el puntaje: {e}")

            messagebox.showerror("Error", "Error al actualizar el puntaje: " + str(e))

    def iniciar\_ruleta(self):

        if not self.girando:  # Evitar que se inicie otra vez mientras está girando

            self.boton\_iniciar.config(state=tk.DISABLED)  # Deshabilitar el botón

            self.speed = self.initial\_speed

            self.girando = True

            self.animar\_flecha()

    def animar\_flecha(self):

        if self.speed > 0:

            self.angle += self.speed

            self.speed -= self.deceleration

            if self.angle >= 360:

                self.angle -= 360

            if self.flecha:

                self.canvas.delete(self.flecha)

            self.flecha = self.dibujar\_flecha(self.angle)

            categoria = self.obtener\_categoria(self.angle)

            self.label\_categoria.config(text=f"Categoria: {categoria}")

            self.root.after(50, self.animar\_flecha)

        else:

            self.girando = False

            self.finalizar\_ruleta()

    def finalizar\_ruleta(self):

        categoria = self.obtener\_categoria(self.angle)

        self.label\_categoria.config(text=f"Categoría seleccionada: {categoria}")

        self.boton\_iniciar.config(state=tk.NORMAL)  # Habilitar el botón nuevamente

        self.mostrar\_pregunta(categoria)

    def mostrar\_pregunta(self, categoria):

        try:

            pregunta, opciones, respuesta\_correcta = self.obtener\_pregunta\_y\_opciones(categoria)

            if pregunta:

                self.mostrar\_ventana\_respuesta(pregunta, opciones, respuesta\_correcta)

            else:

                messagebox.showerror("Error", "No se pudo obtener la pregunta.")

        except Exception as e:

            messagebox.showerror("Error", f"Error al obtener la pregunta: {e}")

    def obtener\_pregunta\_y\_opciones(self, categoria):

        archivo = f"preguntas{self.categorias.index(categoria)}.txt"

        try:

            with open(archivo, 'r', encoding="utf-8") as file:

                preguntas = json.load(file)

                pregunta\_data = random.choice(preguntas)

                pregunta = pregunta\_data["pregunta"]

                opciones = list(pregunta\_data["opciones"].values())

                respuesta\_correcta = pregunta\_data["respuesta\_correcta"]

                return pregunta, opciones, respuesta\_correcta

        except FileNotFoundError:

            messagebox.showerror("Error", f"No se encontró el archivo {archivo}.")

            return None, None, None

    def mostrar\_ventana\_respuesta(self, pregunta, opciones, respuesta\_correcta):

        self.ventana\_respuesta = tk.Toplevel(self.root)

        self.ventana\_respuesta.title("Pregunta")

        self.ventana\_respuesta.geometry("400x300")

        label\_pregunta = tk.Label(self.ventana\_respuesta, text=pregunta, font=("Arial", 14), wraplength=350)

        label\_pregunta.pack(pady=20)

        opciones\_con\_letras = ['a', 'b', 'c', 'd']

        for i, opcion in enumerate(opciones):

            boton\_opcion = tk.Button(self.ventana\_respuesta, text=f"{opciones\_con\_letras[i].upper()}. {opcion}", font=("Arial", 12), command=lambda opt=opciones\_con\_letras[i]: self.elegir\_respuesta(opt, respuesta\_correcta))

            boton\_opcion.pack(pady=5)

    def elegir\_respuesta(self, opcion, respuesta\_correcta):

        if opcion == respuesta\_correcta:

            messagebox.showinfo("Respuesta Correcta", "¡Correcto! Has ganado 10 puntos.")

            self.actualizar\_puntaje(10)

        else:

            messagebox.showinfo("Respuesta Incorrecta", "Respuesta incorrecta. No se sumaron puntos.")

        self.ventana\_respuesta.destroy()

        print("Regresando a la ruleta...")

    def obtener\_categoria(self, angle):

        index = int((angle % 360) / (360 / len(self.categorias)))

        return self.categorias[index]

    def dibujar\_flecha(self, angle):

        radianes = math.radians(angle)

        x1 = self.center\_x + self.flecha\_radius \* math.cos(radianes)

        y1 = self.center\_y + self.flecha\_radius \* math.sin(radianes)

        return self.canvas.create\_line(self.center\_x, self.center\_y, x1, y1, width=3, fill="red")

    def ver\_puntaje(self):

        messagebox.showinfo("Puntaje Actual", f"Tu puntaje es: {self.puntaje}")

    def cerrar\_sesion(self):

        try:

            mensaje = closeSession(self.base\_url, self.username, self.password)

            if "Sseccion cerrada":

                self.root.quit()

            else:

                messagebox.showerror("Error", mensaje)

        except Exception as e:

            messagebox.showerror("Error", f"Error al cerrar sesión: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = Ruleta(root, "usuario", "contraseña", Base\_Url)

    root.mainloop()

**RESULTADOS:**

La ruleta animada funciona correctamente, mostrando una interfaz gráfica interactiva que selecciona categorías del juego de manera aleatoria. El giro se realiza con una flecha con animaciones fluidas, y al detenerse, identifica la categoría según el ángulo final, cargando preguntas relacionadas de archivos de **preguntas.txt**. Las preguntas se presentan en una nueva ventana junto con opciones para responder, evaluando respuesta y actualizando el puntaje del jugador en el archivo local, además que puede cerrar sesión sin problemas.

**PROCEDIMIENTO DE LA INTERFAZ**

El archivo **interfaz\_KingKizz.py** crea una ventana principal para la gestión de usuarios con opciones de registro, inicio de sesión y visualización de usuarios conectados. En el registro, verifica si el usuario ya existe en el archivo usuarios.txt antes de agregar un nuevo registro. En el inicio de sesión, valida las credenciales y cambia el estado del usuario a **conectado** en el archivo **usuarios.txt.** Si el inicio es exitoso, se abre una ventana secundaria que carga la ruleta definida en ruleta.py. También incluye un método para mostrar usuarios conectados usando **getList,** pero esta función no opera debido a problemas de conectividad con el servidor. El sistema funciona de manera efectiva para un entorno local, pero carece de sincronización entre múltiples computadores debido a la falta de conexión por IP.

**CÓDIGO DE LA INTERFAZ**

import tkinter as tk

from tkinter import messagebox

from PIL import Image, ImageTk

from trivia\_client import openSession, getList

from ruleta import Ruleta

Usuarios\_File = "usuarios.txt"

Base\_Url = "http://localhost:80"

def cargar\_usuarios():

    """

    Carga los usuarios del archivo `usuarios.txt` y los devuelve como una lista de diccionarios.

    Filtra líneas vacías o mal formateadas.

    """

    usuarios = []

    try:

        with open(Usuarios\_File, "r", encoding="utf-8") as file:

            for linea in file:

                datos = linea.strip().split(",")

                if len(datos) == 4:  # Asegurarse de que la línea tenga el formato correcto

                    nombre, contrasena, puntaje, estado = datos

                    usuarios.append({

                        "nombre": nombre,

                        "contrasena": contrasena,

                        "puntaje": int(puntaje),

                        "estado": estado

                    })

    except FileNotFoundError:

        # Si el archivo no existe, lo creamos vacío

        with open(Usuarios\_File, "w", encoding="utf-8") as file:

            pass

    return usuarios

def guardar\_usuarios(usuarios):

    """

    Guarda los usuarios en el archivo `usuarios.txt`.

    """

    with open(Usuarios\_File, "w", encoding="utf-8") as file:

        for usuario in usuarios:

            file.write(f"{usuario['nombre']},{usuario['contrasena']},{usuario['puntaje']},{usuario['estado']}\n")

class VentanaPrincipal(tk.Tk):

    def \_\_init\_\_(self):

        """

        Ventana principal para manejar registro, inicio de sesión y mostrar usuarios conectados.

        """

        super().\_\_init\_\_()

        self.title("Gestión de Sesión")

        self.geometry("400x400")

        # Cargar imagen de fondo

        self.canvas = tk.Canvas(self, width=400, height=400)

        self.canvas.pack(fill="both", expand=True)

        self.cargar\_fondo("le1.jpeg")

        # Componentes de la interfaz

        tk.Label(self, text="Usuario", font=("Arial", 14), bg="white").place(x=120, y=100)

        self.entry\_usuario = tk.Entry(self, font=("Arial", 12))

        self.entry\_usuario.place(x=120, y=130, width=160)

        tk.Label(self, text="Contraseña", font=("Arial", 14), bg="white").place(x=120, y=160)

        self.entry\_contrasena = tk.Entry(self, font=("Arial", 12), show="\*")

        self.entry\_contrasena.place(x=120, y=190, width=160)

        tk.Button(self, text="Registrar Usuario", font=("Arial", 12), command=self.registrar\_usuario).place(x=120, y=230, width=160)

        tk.Button(self, text="Iniciar Sesión", font=("Arial", 12), command=self.iniciar\_sesion).place(x=120, y=270, width=160)

        tk.Button(self, text="Ver Usuarios Conectados", font=("Arial", 12), command=self.mostrar\_usuarios).place(x=120, y=310, width=160)

    def cargar\_fondo(self, ruta\_imagen):

        """

        Carga una imagen de fondo para la ventana principal.

        """

        try:

            imagen = Image.open(ruta\_imagen)

            imagen = imagen.resize((400, 400), Image.Resampling.LANCZOS)

            self.fondo = ImageTk.PhotoImage(imagen)

            self.canvas.create\_image(0, 0, image=self.fondo, anchor="nw")

        except Exception as e:

            messagebox.showerror("Error", f"No se pudo cargar la imagen de fondo: {e}")

    def registrar\_usuario(self):

        """

        Registra un nuevo usuario en `usuarios.txt` si no existe.

        """

        usuario = self.entry\_usuario.get().strip()

        contrasena = self.entry\_contrasena.get().strip()

        if not usuario or not contrasena:

            messagebox.showwarning("Campos Vacíos", "Debe llenar todos los campos.")

            return

        usuarios = cargar\_usuarios()

        if any(u["nombre"] == usuario for u in usuarios):

            messagebox.showerror("Error", "El usuario ya existe.")

            return

        usuarios.append({"nombre": usuario, "contrasena": contrasena, "puntaje": 0, "estado": "desconectado"})

        guardar\_usuarios(usuarios)

        messagebox.showinfo("Registro", "Usuario registrado exitosamente.")

        self.limpiar\_campos()

    def iniciar\_sesion(self):

        """

        Inicia sesión verificando el usuario y contraseña con `trivia\_client` y actualiza el archivo local.

        """

        usuario = self.entry\_usuario.get().strip()

        contrasena = self.entry\_contrasena.get().strip()

        if not usuario or not contrasena:

            messagebox.showwarning("Campos Vacíos", "Debe llenar todos los campos.")

            return

        try:

            respuesta = openSession(Base\_Url, usuario, contrasena)

            if "iniciada" in respuesta.lower():

                usuarios = cargar\_usuarios()

                for u in usuarios:

                    if u["nombre"] == usuario:

                        u["estado"] = "conectado"

                        guardar\_usuarios(usuarios)

                        break

                messagebox.showinfo("Inicio de Sesión", "Sesión iniciada exitosamente.")

                self.limpiar\_campos()

                self.abrir\_ventana\_ruleta(usuario, contrasena)

            else:

                messagebox.showerror("Error", respuesta)

        except Exception as e:

            messagebox.showerror("Error", f"No se pudo iniciar sesión: {e}")

    def abrir\_ventana\_ruleta(self, username, password):

        """

        Abre la ventana de la ruleta tras iniciar sesión correctamente.

        """

        ventana\_ruleta = tk.Toplevel(self)

        ventana\_ruleta.geometry("700x800")

        ventana\_ruleta.title("Ruleta de Trivia")

        # Crear instancia de Ruleta con los argumentos requeridos

        Ruleta(ventana\_ruleta, username, password, Base\_Url)

    def mostrar\_usuarios(self):

        """

        Muestra una lista de usuarios conectados obtenidos con `trivia\_client`.

        """

        try:

            usuario = self.entry\_usuario.get().strip()

            contrasena = self.entry\_contrasena.get().strip()

            if not usuario or not contrasena:

                messagebox.showwarning("Campos Vacíos", "Debe llenar todos los campos.")

                return

            respuesta = getList(Base\_Url, usuario, contrasena)

            if respuesta:

                messagebox.showinfo("Usuarios Conectados", f"Usuarios conectados:\n{respuesta}")

            else:

                messagebox.showinfo("Usuarios Conectados", "No hay usuarios conectados en este momento.")

        except Exception as e:

            messagebox.showerror("Error", f"No se pudo obtener la lista de usuarios: {e}")

    def limpiar\_campos(self):

        """

        Limpia los campos de entrada de usuario y contraseña.

        """

        self.entry\_usuario.delete(0, tk.END)

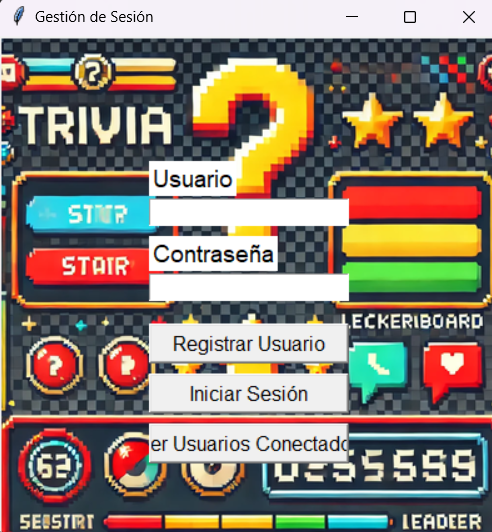
        self.entry\_contrasena.delete(0, tk.END)

if \_\_name\_\_ == "\_\_main\_\_":

    app = VentanaPrincipal()

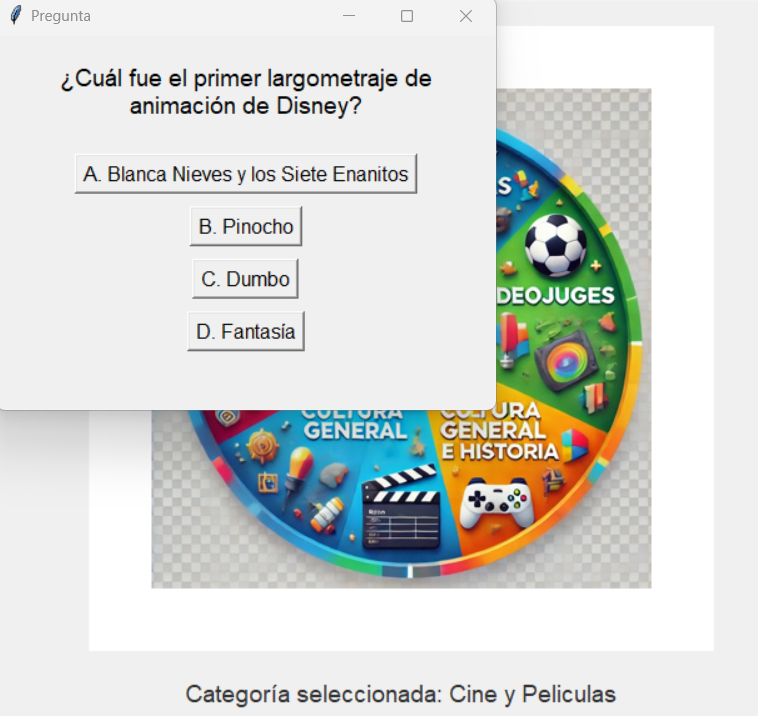
    app.mainloop()

**RESULTADOS:** La interfaz principal permite registrar nuevos usuarios en el archivo local **usuarios.txt**, verificando duplicados y almacenando sus datos correctamente. El inicio de sesión funciona validando credenciales en el archivo local y actualizando el estado del usuario a **conectado.** Tras un inicio exitoso, se abre la ruleta para jugar. No obstante, la interfaz falla al intentar sincronizar usuarios y sesiones con otros computadores debido a problemas de conexión con el servidor por IP, lo que restringe la funcionalidad distribuida del sistema.

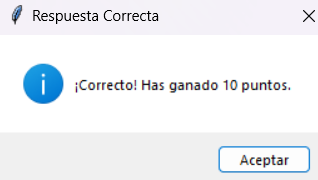


1. **PRUEBAS DE ESCRITORIO**

Se inicia la ruleta con el botón **inicia ruleta** y se detiene en una categoría al azar y da una pregunta



Nos da una ventana emergente, si la respuesta es correcta suma 10 puntos, si la respuesta es incorrecta no la se suman puntos, estas se comparan con las respuestas del juego.



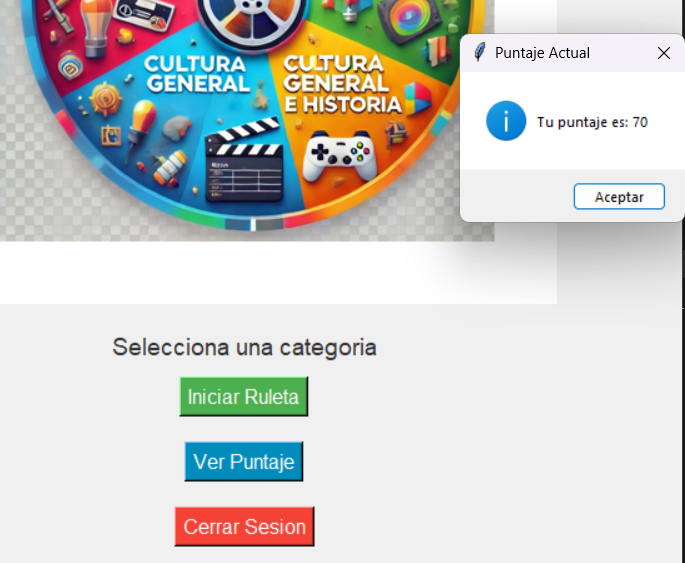
Y se suman al archivo **usuario.txt** en el nombre del jugador.

**Antes**



**Después**

****

**Ver puntaje:** Se presiona el botón de ver puntaje y aparece un mensaje emergente del puntaje que se va actualizando

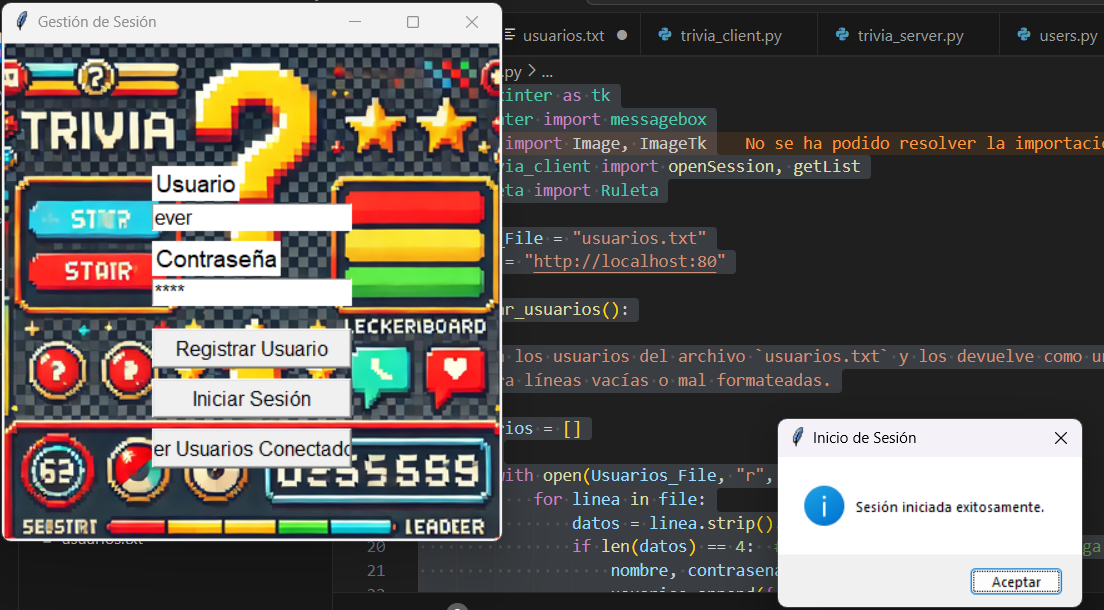
Cierre de sesión, cuando se le da al boton **cerrar sesión** se cierra automaticamente cambiando el archivo de **usurios.txt**

**Antes**

****

**Después**

****

****

**Antes**

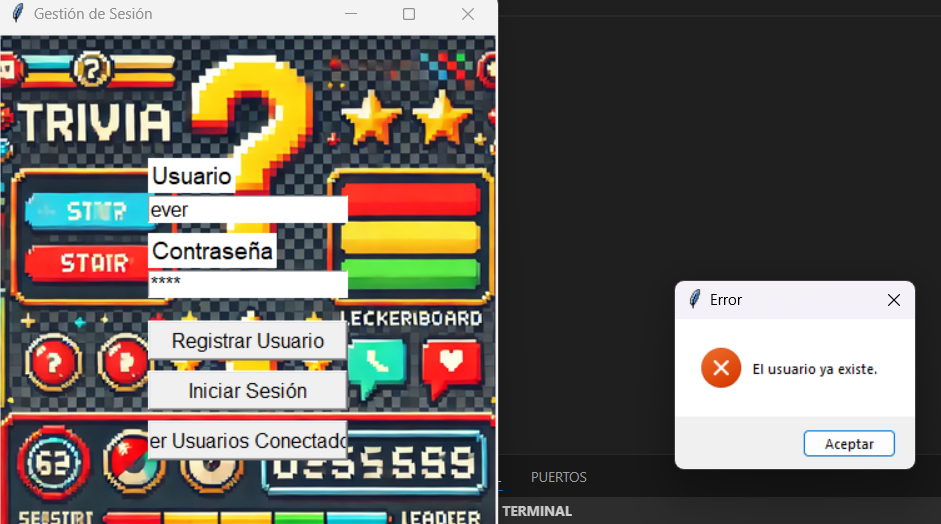
****

**Después**

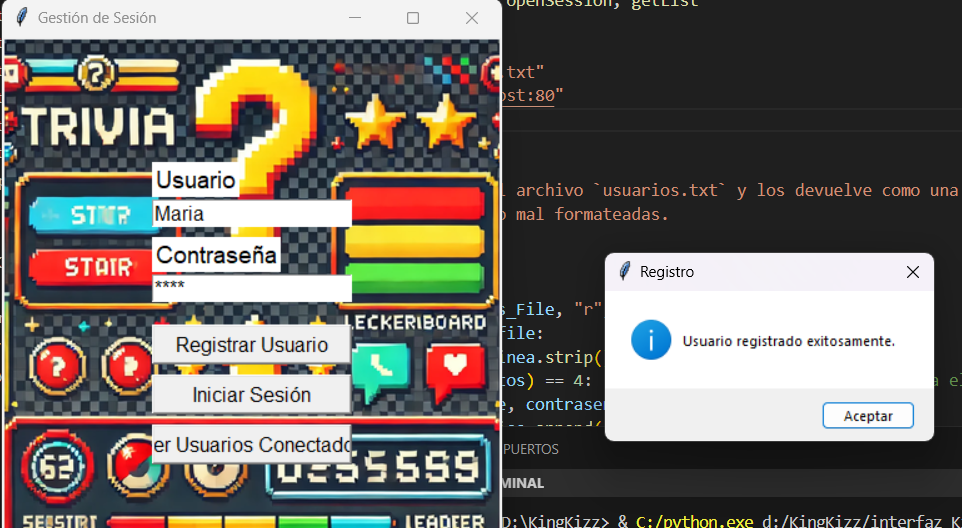
****

**Manejo de duplicado:** Cuándo el usuario ya está registrado

**Prueba:** Usuario con el mismo nombre, pero diferente contraseña



**Registro de otro usuario:** Se registra el usuario de manera correcta y aparece en el archivo **usuarios.txt**



****

1. **PRUEBAS UNITARIAS**

Se verificó que la función **cargar\_usuarios()** pueda leer correctamente los datos de los usuarios desde el archivo usuarios.txt, gestionando adecuadamente líneas mal formadas o vacías.

**CÓDIGO**

import unittest

from unittest.mock import patch, mock\_open

from interfaz\_KingKizz import cargar\_usuarios

class TestInterfazKingKizz(unittest.TestCase):

    @patch("builtins.open", new\_callable=mock\_open, read\_data="user1,pass1,100,conectado\nuser2,pass2,200,desconectado\n")

    def test\_cargar\_usuarios(self, mock\_file):

        usuarios = cargar\_usuarios()

        self.assertEqual(len(usuarios), 2)

        self.assertEqual(usuarios[0]["nombre"], "user1")

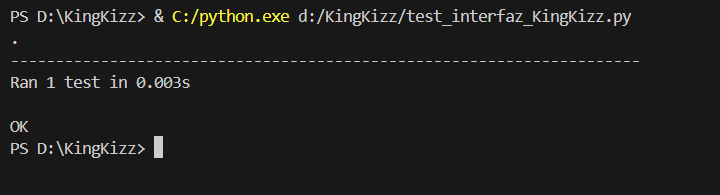
        self.assertEqual(usuarios[0]["puntaje"], 100)

        self.assertEqual(usuarios[1]["estado"], "desconectado")

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

**RESULATDO:** La función cargó exitosamente dos usuarios con el formato correcto (nombre, puntaje, estado), y no presentó errores al manejar líneas mal formadas o archivos vacíos.

**Estado:** Aprobado.

**PRUEBA:** Se probó que la función **guardar\_usuarios()** escriba correctamente la lista de usuarios en el archivo **usuarios.txt.**

**CÓDIGO**

import unittest

from unittest.mock import patch, mock\_open

from interfaz\_KingKizz import guardar\_usuarios

class TestInterfazKingKizz(unittest.TestCase):

    @patch("builtins.open", new\_callable=mock\_open)

    def test\_guardar\_usuarios(self, mock\_file):

        usuarios = [

            {"nombre": "user1", "contrasena": "pass1", "puntaje": 100, "estado": "conectado"},

            {"nombre": "user2", "contrasena": "pass2", "puntaje": 200, "estado": "desconectado"}

        ]

        guardar\_usuarios(usuarios)

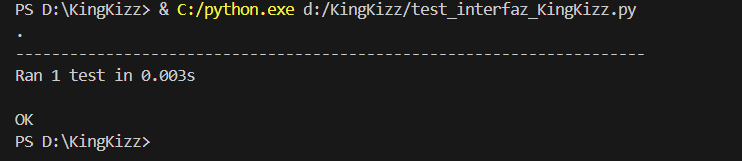
        mock\_file().write.assert\_any\_call("user1,pass1,100,conectado\n")

        mock\_file().write.assert\_any\_call("user2,pass2,200,desconectado\n")

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

**RESULATADO:** Los datos de los usuarios se guardaron correctamente en el archivo, respetando el formato y asegurando que cada usuario apareciera en una línea separada.



1. **RESULTADOS ESPERADOS Y RESULTADOS**

* **Registro de Usuarios:** Se esperaba que el registro de usuarios permitiera crear cuentas no solo localmente, sino también sincronizarlas entre diferentes computadoras a través de una conexión por IP. Esto permitiría que los usuarios registrados en un equipo fueran accesibles en otros equipos de forma remota. Sin embargo, en los resultados obtenidos, la funcionalidad de registro local funciona correctamente y los usuarios se guardan en el archivo **usuarios.txt,** pero no se logra la sincronización entre equipos. La falta de una conexión efectiva por IP impide que los registros sean compartidos entre dispositivos, lo que limita la capacidad de acceso distribuido a los usuarios.
* **Inicio de Sesión:** El inicio de sesión debería validar las credenciales de los usuarios, ya sea desde el archivo local **usuarios.txt** o desde un servidor centralizado, permitiendo la autenticación distribuida. En las pruebas realizadas, el inicio de sesión local funciona correctamente, permitiendo a los usuarios iniciar sesión con sus credenciales almacenadas localmente. Sin embargo, la conexión entre equipos no se estableció correctamente, lo que impide el inicio de sesión distribuido entre computadoras, limitando la funcionalidad de acceso remoto a los sistemas.
* **Ruleta para Selección de Categoría:** La funcionalidad de la ruleta, que debía seleccionar aleatoriamente una categoría para el jugador, se comporta como se esperaba. La animación de la ruleta y la selección de la categoría se ejecutan sin problemas, mostrando una interacción fluida para el usuario. La ruleta funciona correctamente en cuanto a la mecánica de selección, cumpliendo con los objetivos establecidos en cuanto a la dinámica del juego.
* **Mostrar Preguntas Según Categoría:** La funcionalidad que carga las preguntas desde archivos locales y las muestra según la categoría seleccionada se ejecuta de manera correcta. Las preguntas se cargan sin inconvenientes, y se muestra adecuadamente la información al jugador. La evaluación de las respuestas es precisa, mostrando si la respuesta es correcta o incorrecta y gestionando adecuadamente el flujo del juego
* **Actualización de Puntaje:** El puntaje se actualiza correctamente de forma local en el archivo usuarios.txt cada vez que el jugador responde correctamente a una pregunta. Sin embargo, la sincronización entre equipos no funciona correctamente debido a la desconexión por IP. Esto significa que, aunque el puntaje se actualiza en un equipo, no se refleja en otros equipos, lo que impide una gestión centralizada del puntaje entre dispositivos.
* **Gestión de Usuarios Conectados:** Se esperaba que la función de gestión de usuarios conectados pudiera mostrar una lista en tiempo real de los usuarios activos, obtenida desde el servidor. Sin embargo, debido a la falta de conexión por IP, no fue posible obtener esta información, lo que impide que se muestre una lista actualizada de los usuarios conectados entre diferentes equipos. Esto limita la capacidad de interacción en tiempo real entre los usuarios en la plataforma.

1. **VARIABLES, CLASES Y MÉTODOS**

* **Variables**

En el archivo **interfaz\_KingKizz.py**, las principales variables son **Usuarios\_File**, que contiene el nombre del archivo donde se almacenan los datos de los usuarios (usuarios.txt), y **Base\_Url,** que define la URL base utilizada para interactuar con el servidor. En el archivo **ruleta.py**, las variables clave incluyen puntaje, que almacena el puntaje actual del jugador, y categorías, que es una lista de las diferentes categorías disponibles para las preguntas del juego (deportes, música, cine, etc.).

**Excepciones**

En **interfaz\_KingKizz.py**, se manejan excepciones como **FileNotFoundError** en el método **cargar\_usuarios()** para crear el archivo usuarios.txt si no se encuentra. También se manejan excepciones generales en los métodos de inicio de sesión y gestión de usuarios, donde se capturan errores de conexión a la red o fallos en la validación de credenciales. En **ruleta.py**, se capturan excepciones en el método **obtener\_pregunta\_y\_opciones()** si no se encuentra el archivo de preguntas correspondiente, utilizando un bloque **try-except** para manejar el error y notificar al usuario si el archivo no se puede cargar.

**Clases**

En **interfaz\_KingKizz.py**, la clase principal es **VentanaPrincipal**, que gestiona la ventana de la interfaz gráfica de usuario. Esta clase es responsable de las interacciones del usuario, como el registro, inicio de sesión y la visualización de usuarios conectados. En **ruleta.py**, la clase principal es **Ruleta**, que se encarga de la lógica del juego de la ruleta. Maneja la animación de la ruleta, la selección aleatoria de categorías, y las interacciones del jugador con las preguntas y respuestas, además de actualizar el puntaje del jugador en el servidor.

**Métodos**

En **interfaz\_KingKizz.py**, los métodos principales incluyen cargar\_usuarios(), que lee los datos de los usuarios desde el archivo usuarios.txt, y guardar\_usuarios(), que escribe los datos actualizados en el mismo archivo. Los métodos **registrar\_usuario()** e **iniciar\_sesion()** gestionan el registro y la validación de las credenciales del usuario. El método **mostrar\_usuarios()** obtiene la lista de usuarios conectados desde el servidor y la muestra en un cuadro de diálogo. En **ruleta.py**, los métodos principales de la clase **Ruleta** incluyen **iniciar\_ruleta(),** que inicia la animación y el giro de la ruleta; **actualizar\_puntaje(),** que actualiza el puntaje en el servidor; y **elegir\_respuesta(),** que evalúa si la respuesta del jugador es correcta o incorrecta. También está **obtener\_pregunta\_y\_opciones(),** que carga las preguntas correspondientes a la categoría seleccionada.

1. **CONCLUSIONES**

 El sistema funciona localmente, pero no se sincroniza entre equipos debido a fallos en la conexión por IP.

 La ruleta, preguntas y actualización de puntaje operan correctamente en un solo computador.

 La sincronización de usuarios y puntajes no funciona sin conexión al servidor por IP.

 La interfaz gráfica es fácil de usar y proporciona una experiencia de juego fluida.

 Es necesario solucionar los problemas de conexión por IP para permitir la sincronización entre múltiples dispositivos.

1. **REFERENCIAS**

En este laboratorio no se usan ya que no tuvimos una guía como tal, no se hacen con ayuda de videos y las imágenes de la ruleta y el fondo son hechas con IA, pero cabe en recalcar que se tuvo apoyo del profesor Alexander Parrado y el profesor Julián Barrero