Carpeta Proyecto: GRUPO-1—TAXI

Subcarpeta: Graficos

Todos los archivos de código abajo están dentro de la carpeta de proyecto GRUPO-1—TAXI

Archivo: Check_passwords.py

```
import pandas as pd
import hashlib
import tkinter as tk
from IU_Class import init_game

#Cuando tengamos una base de datos como tal
#datos_usuarios = pd.read_csv("usuarios.csv")


def LogIn(username, password):
    datos_usuarios = pd.read_csv("Usuarios.csv")
    password_inp = hashlib.sha256(password.encode('utf-8')).hexdigest()
    password_local = datos_usuarios.loc[datos_usuarios["Usuarios"] ==
username]["Passwords"].item()

    if username not in datos_usuarios["Usuarios"].values or password_inp !=
password_local:
        tk.messagebox.showinfo(title = "Error", message = "Usuario o
contraseña incorrecta")

    else:
        init_game(username)

def Register(username, password, s_quest, s_answer):
    datos_usuarios = pd.read_csv("Usuarios.csv")
    if datos_usuarios.Usuarios.isin([username]).any():
        tk.messagebox.showinfo(title = "Error", message = "Nombre de usuario
en uso, por favor eliga otro")
    else:
        password_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()
        df = pd.DataFrame({'Usuarios' : [username], 'Passwords':
[password_hash], 'Pregunta Secreta' : s_quest, 'Respuesta Secreta': s_answer})
        datos_usuarios = pd.concat([datos_usuarios, df], ignore_index = True)
        datos_usuarios.to_csv('Usuarios.csv', index = False)
        tk.messagebox.showinfo(title = "Registro completado", message =
"Registo completado")
        return(True)
```

```
def Pregunta(username):
    datos_usuarios = pd.read_csv("Usuarios.csv")
    if not datos_usuarios.Usuarios.isin([username]).any():
        tk.messagebox.showinfo(title = "Error", message = "Usuario no
encontrado")
    else:
        pregunta_s = datos_usuarios.loc[datos_usuarios["Usuarios"] ==
username]["Pregunta Secreta"].item()
        tk.messagebox.showinfo(title = "Pregunta", message = pregunta_s)

def Respuesta(username, answer, new_pswd):
    datos_usuarios = pd.read_csv("Usuarios.csv")
    local_answ = datos_usuarios.loc[datos_usuarios["Usuarios"] ==
username]["Respuesta Secreta"].item()
    if local_answ == answer:
        new_pswd_hash = hashlib.sha256(new_pswd.encode('utf-8')).hexdigest()
        datos_usuarios.loc[datos_usuarios["Usuarios"] == username,
"Passwords"] = new_pswd_hash
        datos_usuarios.to_csv('Usuarios.csv', index = False)
        tk.messagebox.showinfo(title = "Exito", message = "Contraseña
cambiada")
    else:
        tk.messagebox.showinfo(title = "Error", message = "Respuesta
Incorrecta")
```

Archivo: Clases taxi.py

```
'''
Definimos una clase principal conductor donde incluimos como atributos las
tarifas estándar para todos los vehiculos tipo taxi.
Después con herencias, establecemos dos clases, VTC (uber, cabify, etc.) que
heredan las tarifas de conductor pero que permiten establecer atributos
distintos.
En el caso de los VTC, como quieren competir con los taxis, establecen el
descuento que consideren necesario a sus tarifas, tanto en parado como en
movimiento, pasando como argumentos los porcentajes de descuento para cada uno
de ellos. Por defecto no hacen descuento.
En el caso de los taxi, su tarifa es fija, sin embargo por la noche cobran
plus de nocturnidad, por lo tanto, cada taxista puede ajustar cuanto quiere
cobrar extra por la noche. Para ello el primer argumento define si está en
horario nocturno mediante True o False, y de ser noche, establece el
procenatje (que por defecto es 0) que se debe sumar a la tarifa total.
'''

class conductor:
    tarifa_parado = 0.02
```

```
    tarifa_movimiento = 0.05

class VTC(conductor):
    def __init__(self, descuento_p = 0, descuento_m = 0):
        self.tarifa_movimiento += self.tarifa_movimiento * (descuento_m/100)
        self.tarifa_parado += self.tarifa_parado * (descuento_p/100)
        self.tarifa_movimiento = round(self.tarifa_movimiento, 2)
        self.tarifa_parado = round(self.tarifa_parado, 2)

class taxista(conductor):
    def __init__(self, noche, prcnt = 0):
        if noche:
            self.tarifa_parado += self.tarifa_parado * (prcnt/100)
            self.tarifa_movimiento += self.tarifa_movimiento * (prcnt/100)
            self.tarifa_parado = round(self.tarifa_parado, 2)
            self.tarifa_movimiento = round(self.tarifa_movimiento, 2)


taxi = taxista(True, 20)
taxi.tarifa_parado
uber = VTC(20, 30)
uber.tarifa_parado
```

Archivo: Creador de logs.py

```
import logging # Biblioteca para generar los logs

def creador_logs():
    '''
    Se crea una función que va a iniciar el proceso de logear los sucesos.
    getLogger es una función de la biblioteca logging, a la que le asignamos
__name__ que asigna a la variable el nombre del modulo a emplear
    basicConfig permite configurar los logs que se van a producir
    Por último devemos devolver el logger que hemos creado
    '''
    logger = logging.getLogger(__name__)
    logging.basicConfig(filename = 'archivo_logs.log', #nombre del archivo
                        level = logging.INFO,  #nivel mínimo para el que va a
generarse logs (mirar tabla
https://docs.python.org/3/library/logging.html#levels)
                        format='%(asctime)s - %(levelname)s - %(message)s',
#formato de presentación del log, asctime será la hora, levelname el nombre
del nivel del mensaje y message el mensaje que nosotros asignemos
                        datefmt='%d/%m/%Y %I:%M:%S: %p') #formato de fecha,
puesto para día, mes, año, hora, minuto, segundo y AM o PM.
    return logger
```

```
def taximetro():
    logger = creador_logs() #llamamos a la función previa
    '''
    El código a continuación es meramente una prueba para ver como funcinaría
el logger
    Los diferentes niveles se establecen con un .level(.info/.error) que
saldrán en el archivo del log
    '''
    logger.info("Inicio del log")
    movimiento = 0.05
    parado = 0.02
    try: # Para guardar los errores utilizar un try/except
        movimiento/0
        logger.info("Funcionando")
        logger.info("Final")
    except ZeroDivisionError:
        logger.error("Se ha producido un error")

taximetro()
```

Archivo: Estructura base.py

```
import time

class conductor:
    tarifa_parado = 0.02
    tarifa_movimiento = 0.05

class VTC(conductor):
    def __init__(self, descuento_p = 0, descuento_m = 0):
        self.tarifa_movimiento += self.tarifa_movimiento * (descuento_m/100)
        self.tarifa_parado += self.tarifa_parado * (descuento_p/100)
        self.tarifa_movimiento = round(self.tarifa_movimiento, 2)
        self.tarifa_parado = round(self.tarifa_parado, 2)

class taxista(conductor):
    def __init__(self, noche, prcnt = 0):
        if noche:
            self.tarifa_parado += self.tarifa_parado * (prcnt/100)
            self.tarifa_movimiento += self.tarifa_movimiento * (prcnt/100)
            self.tarifa_parado = round(self.tarifa_parado, 2)
            self.tarifa_movimiento = round(self.tarifa_movimiento, 2)


def tipo():
    #LogIn()
    conductor = input("Indica 'Taxista' o 'VTC'")
    if conductor.lower() == 'taxista':
```

```
        noche = input('¿Es de noche?(si/no)')
        if noche.lower() == 'si':
            tasa = input('Indica el procentaje extra de tarifa nocturna')
            mov = taxista(True, int(tasa))
        else:
            mov = taxista(False)
    else:
        desc_mov = input("Indica la tasa de descuento en movimiento")
        desc_par = input("Indica la tasa de descuento en parado")
        mov = VTC(int(desc_mov), int(desc_par))
    return(mov)

def precio(movimiento):
    tasas = tipo()
    precio = 0
    taximetro = True
    while taximetro and precio <= 1:
        if movimiento:
            precio += tasas.tarifa_movimiento
            time.sleep(1)
            print(precio)
        else:
            precio += tasas.tarifa_parado
            time.sleep(1)
            print(precio)
    return(round(int(precio), 2))
```

Archivo: GUI Entrada.py

```
import tkinter as tk
import pandas as pd
import hashlib
from tkinter import messagebox
from Check_passwords import LogIn, Register, Pregunta, Respuesta

class GUI:
    def __init__(self):
        self.root = tk.Tk()
        self.title = self.root.title("Inicio de Sesion")
        self.root.geometry('500x500')
        self.root.configure(bg = '#824AB5')

        self.inicio = tk.Button(self.root, text = "Iniciar Sesion", font =
('Lucida Console', 16), command = self.login_screen)
        self.inicio.pack()
```

```
        self.reg = tk.Button(self.root, text = "Registrarse", font = ('Lucida
Console', 16), command = self.reg_screen)
        self.reg.pack()

        self.root.mainloop()

    def login_screen(self):
        for widget in self.root.winfo_children():
            widget.destroy()

        self.label = tk.Label(self.root, text = "Inicie Sesion", font =
('Lucida Console', 16))

        self.user = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.user.pack()

        self.password = tk.Entry(self.root, font =('Lucida Console', 16), show
= '*')
        self.password.pack()

        self.button = tk.Button(self.root, text = "Iniciar Sesion", font =
('Lucida Console', 16), command = self.check_password)
        self.button.pack()

        self.button = tk.Button(self.root, text = "Olvidé mi contraseña", font
= ('Lucida Console', 16), command = self.res_pswd)
        self.button.pack()

    def reg_screen(self):
        for widget in self.root.winfo_children():
            widget.destroy()

        self.label = tk.Label(self.root, text = 'Registro', font = ('Lucida
Console', 16))
        self.user = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.user.pack()

        self.password = tk.Entry(self.root, font = ('Lucida Console', 16),
show = '*')
        self.password.pack()

        self.quest = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.quest.pack()

        self.answ = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.answ.pack()
```

```
        self.button = tk.Button(self.root, text = "Registro", font = ('Lucida
Console', 16), command = self.register)
        self.button.pack()

    def res_pswd(self):
        for widget in self.root.winfo_children():
            widget.destroy()

        self.label = tk.Label(self.root, text = "Reiniciar contraseña", font =
('Lucida Console', 16))
        self.label.pack()

        self.user = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.user.pack()

        self.quest = tk.Button(self.root, text = "Ver pregunta secreta", font
= ('Lucida Console', 16), command = self.get_quest)
        self.quest.pack()

        self.label2 = tk.Label(self.root, text = "Respuesta secreta", font =
('Lucida Console', 16))
        self.label2.pack()

        self.answer = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.answer.pack()

        self.label3 = tk.Label(self.root, text = "Nueva contraseña", font =
('Lucida Console', 16))
        self.label3.pack()

        self.new_pswd = tk.Entry(self.root, font = ('Lucida Console', 16))
        self.new_pswd.pack()

        self.submit = tk.Button(self.root, text = "Enviar", font = ('Lucida
console', 16), command = self.change_pswd)
        self.submit.pack()

        self.back = tk.Button(self.root, text = "Atrás", font = ('Lucida
Console', 16), command = self.login_screen)
        self.back.pack()


    def check_password(self):
        LogIn(self.user.get().lower(), self.password.get())

    def register(self):
         if Register(self.user.get().lower(), self.password.get(),
self.quest.get(), self.answ.get().lower()) == True:
```

```
            self.login_screen()

    def get_quest(self):
        Pregunta(self.user.get().lower())

    def change_pswd(self):
        Respuesta(self.user.get().lower(), self.answer.get().lower(),
self.new_pswd.get())


GUI()
```

Archivo: initApp.py

```
import pandas as pd
import hashlib
import traceback
import datetime

# Especifica la ruta del archivo CSV donde se almacenan los datos de los
usuarios.
path = "Usuarios.csv"

def register_action(mensaje):
# Función para registrar acciones y errores.
# Registra mensajes de acciones y errores en un archivo de registro
(registro.txt) con una marca de tiempo.
    with open("registro.txt", "a") as archivo:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        archivo.write(f"{timestamp} - {mensaje}\n")

def load_data():
# Función que intenta cargar los datos del archivo CSV especificado en path.
# Si el archivo no existe, crea uno nuevo con las columnas Usuarios,
Passwords, Pregunta Secreta y Respuesta Secreta.
    try:
        # Intentar leer el archivo CSV
        datos_usuarios = pd.read_csv(path)
    except FileNotFoundError:
        # Si el archivo no existe, crear uno nuevo
        print(f"El archivo {path} no se encontró. Creando una nueva base de
datos.")
        datos_usuarios = pd.DataFrame(columns=['Usuarios', 'Passwords',
'Pregunta Secreta', 'Respuesta Secreta'])
        datos_usuarios.to_csv(path, index=False)
    except Exception as e:
```

```
        print(f"Ha ocurrido un error inesperado al cargar los datos: {e}")
        datos_usuarios = pd.DataFrame(columns=['Usuarios', 'Passwords',
'Pregunta Secreta', 'Respuesta Secreta'])
    return datos_usuarios

def request_input(mensaje, validacion=None):
# Solicita una entrada del usuario y valida la entrada usando una función de
validación opcional
    while True:
        entrada = input(mensaje).strip()
        if validacion and not validacion(entrada):
            print("La contreña ha de tener como mínimo 8 caracteres. Inténtalo
de nuevo.")
        else:
            return entrada

def validate_password_format(contrasena):
# Valida que la contraseña tenga al menos 8 caracteres
    return len(contrasena) >= 8

def request_password(mensaje):
# Solicita una contraseña del usuario y valida que tenga al menos 8 caracteres
    return request_input(mensaje, validate_password_format)

def register_user(datos_usuarios):
    '''
    Esta función registra un nuevo usuario:
    1. Solicita un nombre de usuario y verifica que no esté en uso.
    2. Solicita una contraseña, pregunta secreta y respuesta secreta.
    3. Guarda los datos del nuevo usuario en el archivo CSV.
    '''
    try:
        print("Iniciando el registro de usuario")
        user = request_input("Escriba un nombre de usuario: ").lower()
        while user in datos_usuarios["Usuarios"].str.lower().values:
            print("Ese nombre de usuario ya está seleccionado.")
            user = request_input("Escriba un nombre de usuario diferente:
").lower()
        password = request_password("Escribe una contraseña segura (mínimo 8
caracteres alfanuméricos): ")
        password_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()
        secret_q = request_input("Escriba una pregunta que solo usted conozca
la respuesta: ").lower()
        secret_a = hashlib.sha256(request_input("Escriba la respuesta a su
pregunta secreta: ").encode('utf-8')).hexdigest()
        nuevo_usuario = pd.DataFrame({'Usuarios': [user], 'Passwords':
[password_hash], 'Pregunta Secreta': [secret_q], 'Respuesta Secreta':
[secret_a]})
```

```
        datos_usuarios = pd.concat([datos_usuarios, nuevo_usuario],
ignore_index=True)
        datos_usuarios.to_csv(path, index=False)
        print("Usuario registrado correctamente.")
        register_action(f"Nuevo usuario registrado: {user}")
    except Exception as e:
        print(f"Ha ocurrido un error al registrar usuario: {e}")
        register_action(f"Error al registrar usuario: {e}")

def validate_password(datos_usuarios, usuario):
    '''
    Esta función valida que la contraseña introducida por el usuario es
correcta.
    En caso que no se introduzca la contraseña correcta en los 6 intentos, se
pide al usuario que reinicie el programa
    '''
    intentos = 0
    password_local = datos_usuarios.loc[datos_usuarios["Usuarios"].str.lower()
== usuario, "Passwords"].values[0]
    while intentos < 6:
        password_inp = request_password("Escriba su contraseña: ")
        password_inp_hash = hashlib.sha256(password_inp.encode('utf-
8')).hexdigest()
        if password_inp_hash == password_local:
            print("Bienvenido")
            register_action(f"Usuario autenticado correctamente: {usuario}")
            return
        else:
            print("Contraseña incorrecta")
            register_action(f"Contraseña incorrecta para usuario: {usuario}")
        intentos += 1
    print("Demasiados intentos fallidos, reinicie el programa")

def change_password(datos_usuarios, usuario):
    '''
    Cambia la contraseña del usuario:
    Verifica la respuesta a la pregunta secreta (hasta 3 intentos).
    Solicita una nueva contraseña que debe ser diferente a la actual.
    '''
    secret_q = datos_usuarios.loc[datos_usuarios["Usuarios"].str.lower() ==
usuario, "Pregunta Secreta"].values[0]
    true_answ = datos_usuarios.loc[datos_usuarios["Usuarios"].str.lower() ==
usuario, "Respuesta Secreta"].values[0]
    intentos = 0

    while intentos < 3:
        secret_answ = hashlib.sha256(request_input(secret_q).encode('utf-
8')).hexdigest()
```

```
        if secret_answ == true_answ:
            current_pass_hash =
datos_usuarios.loc[datos_usuarios["Usuarios"].str.lower() == usuario,
"Passwords"].values[0]
            new_pass = request_password("Introduce tu nueva contraseña (mínimo
8 caracteres alfanuméricos): ")
            new_pass_hash = hashlib.sha256(new_pass.encode('utf-
8')).hexdigest()
            while new_pass_hash == current_pass_hash:
                print("La nueva contraseña no puede ser igual a la actual.")
                new_pass = request_password("Introduce tu nueva contraseña
diferente a la actual: ")
                new_pass_hash = hashlib.sha256(new_pass.encode('utf-
8')).hexdigest()
            datos_usuarios.loc[datos_usuarios["Usuarios"].str.lower() ==
usuario, "Passwords"] = new_pass_hash
            datos_usuarios.to_csv(path, index=False)
            print("Contraseña cambiada")
            register_action(f"Contraseña cambiada para usuario: {usuario}")
            return
        else:
            print("Respuesta incorrecta")
            register_action(f"Respuesta secreta incorrecta para usuario:
{usuario}")
        intentos += 1
    print("Demasiados intentos fallidos, reinicie el programa")
    register_action(f"Demasiados intentos fallidos para cambiar contraseña de
usuario: {usuario}")

def login_user(cambio=False):
    '''
    Gestiona el inicio de sesión o cambio de contraseña:
    Carga los datos de usuarios.
    Si cambio es True, cambia la contraseña; si no, valida la contraseña.
    '''
    datos_usuarios = load_data()
    try:
        usuario = request_input("Escriba su nombre de usuario: ").lower()
        if cambio:
            if usuario not in datos_usuarios["Usuarios"].str.lower().values:
                print("Este usuario no existe")
                register_user(datos_usuarios)
                register_action(f"Intento de cambio de contraseña para usuario
no existente: {usuario}")
            else:
                change_password(datos_usuarios, usuario)
        else:
            if usuario not in datos_usuarios["Usuarios"].str.lower().values:
```

```
                print("Usuario no registrado.")
                register_user(datos_usuarios)
            else:
                validate_password(datos_usuarios, usuario)
    except Exception as e:
        register_action(f"Error: {e}")
        register_action(traceback.format_exc())
        print("Ha ocurrido un error. Por favor, revise el registro.")

def start():
    '''
    Llama a la función login_user con el parámetro cambio
    Si cambio=True para start el proceso de cambio de contraseña al ejecutar
el programa.
    Si cambio=False se inicia el proceso para start sesion en la aplicación
    '''
    #login_user(False)
    login_user(True)

start()
```

Archivo: IU_Class.py

```
import pygame
from sys import exit
import pygame_gui
import time
from datetime import date
import pandas as pd

class Game:
    def __init__(self, user):
        self.FPS = 60
        self.S_Width = 1600
        self.S_Height = 900
        pygame.init()
        self.screen = pygame.display.set_mode((self.S_Width, self.S_Height))
        self.clock = pygame.time.Clock()
        self.manager = pygame_gui.UIManager((self.S_Width, self.S_Height))
        self.user = user

        self.gameStateManager = gameStateManager('start')
        self.start = Start(self.screen, self.gameStateManager)
        self.intro = Intro(self.screen, self.gameStateManager)
        self.taximetro = Taximetro(self.screen, self.gameStateManager)
        self.quit = Quit(self.screen, self.gameStateManager, self.user)
```

```
        self.states = {'start': self.start,
                       'taximetro': self.taximetro,
                       'intro': self.intro,
                       'quit': self.quit}

        self.gameStateManager.set_states(self.states)

    def run(self):
        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.quit.handle_quit()

                # Manejar eventos específicos del estado actual
                self.states[self.gameStateManager.get_state()].handle_events(e
vent)

            self.states[self.gameStateManager.get_state()].run()

            pygame.display.update()
            self.clock.tick(self.FPS)

class Start:
    def __init__(self, display, gameStateManager):
        self.display = display
        self.gameStateManager = gameStateManager

    def handle_events(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_e:
                self.gameStateManager.set_state('level')
        elif event.type == pygame.MOUSEBUTTONDOWN:
            a, b = pygame.mouse.get_pos()
            if self.quit_button_rect.collidepoint((a, b)):
                self.gameStateManager.set_state('quit')
            elif self.login_button_rect.collidepoint((a, b)):
                self.gameStateManager.set_state('intro')

    def run(self):
        # Variables generales
        a, b = pygame.mouse.get_pos()
        login_screen = pygame.image.load('Graficos/base_2.jpeg')
        font = pygame.font.SysFont('Lucida Console', 70)
        color_font = (200, 245, 10, 1)
        color_rect_hover = (91, 23, 202, 0.8)
        color_rect_base = (65, 0, 168, 0.9)
        # Botón Start
        self.login_button_rect = pygame.Rect(500, 400, 650, 80)
```

```
        login_text = font.render('Empezar carrera', True, color_font)
        # Botón Quit
        self.quit_button_rect = pygame.Rect(730, 650, 180, 80)
        quit_text = font.render('Quit', True, color_font)
        self.display.blit(login_screen, (0, 0))
        if self.quit_button_rect.collidepoint((a, b)):
            pygame.draw.rect(self.display, color_rect_hover,
self.quit_button_rect)
        else:
            pygame.draw.rect(self.display, color_rect_base,
self.quit_button_rect)

        if self.login_button_rect.collidepoint((a, b)):
            pygame.draw.rect(self.display, color_rect_hover,
self.login_button_rect)
        else:
            pygame.draw.rect(self.display, color_rect_base,
self.login_button_rect)

        self.display.blit(login_text, (self.login_button_rect.x + 5,
self.login_button_rect.y + 5))
        self.display.blit(quit_text, (self.quit_button_rect.x + 5,
self.quit_button_rect.y + 5))

class Intro:
    def __init__(self, display, gameStateManager):
        self.display = display
        self.gameStateManager = gameStateManager

    def handle_events(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RETURN:
                self.gameStateManager.set_state('taximetro')
                # Pasar el estado actual a Taximetro para iniciar el tiempo
                self.gameStateManager.get_states()['taximetro'].start_time =
time.time()

    def run(self):
        fondo = pygame.image.load('Graficos/base_2.jpeg')
        self.display.blit(fondo, (0, 0))
        texto = pygame.image.load('Graficos/Intro_text (Mediana).png')
        self.display.blit(texto, (100, 100))

class Taximetro:
    def __init__(self, display, gameStateManager):
        self.display = display
        self.gameStateManager = gameStateManager
        self.car = pygame.image.load('Graficos/car1.png')
```

```
        self.car_position = 20
        self.car_mov = False
        self.font = pygame.font.SysFont('Lucida Console', 30)
        self.start_time = None  # Inicializamos start_time como None
        self.score = 0

    def handle_events(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                self.car_mov = not self.car_mov
            elif event.key == pygame.K_p:
                self.gameStateManager.set_state('start')
            elif event.key == pygame.K_RETURN:
                self.gameStateManager.set_state('quit')

    def run(self):
        first_screen = pygame.image.load('Graficos/base_2.jpeg')
        self.display.blit(first_screen, (0, 0))
        color_font = (200, 245, 10, 1)

        if self.start_time is not None:  # Aseguramos que start_time tenga un
valor antes de usarlo
            if self.car_mov:
                self.car_position += 5  # Ajusta la velocidad del coche según
sea necesario
                if self.car_position > 1600:  # 1600 es el ancho de la
pantalla
                    self.car_position = -self.car.get_width()  # Aparecer en
el otro lado
                self.score += 0.05 / 60  # Incrementar la puntuación por
segundo en movimiento
            else:
                self.score += 0.02 / 60  # Incrementar la puntuación por
segundo en parado

            self.display.blit(self.car, (self.car_position, 700))

            # Calcular el tiempo transcurrido en minutos y segundos
            elapsed_time_s = time.time() - self.start_time
            elapsed_minutes = int(elapsed_time_s // 60)
            elapsed_seconds = int(elapsed_time_s % 60)
            clock_text = self.font.render(f'Tiempo:
{elapsed_minutes:02}:{elapsed_seconds:02}', True, (color_font))
            self.display.blit(clock_text, (50, 50))

            # Mostrar la puntuación
            score_text = self.font.render(f'Precio: {round(self.score, 2)} €',
True, (color_font))
```

```
            self.display.blit(score_text, (50, 100))

    def get_score(self):
        return self.score

    def get_total_time(self):
        if self.start_time is None:
            return 0
        return time.time() - self.start_time

class gameStateManager:
    def __init__(self, currentState):
        self.currentState = currentState
        self.states = None  # Inicializamos states como None

    def set_states(self, states):
        self.states = states  # Método para establecer los estados

    def get_states(self):
        return self.states  # Método para obtener los estados

    def get_state(self):
        return self.currentState

    def set_state(self, state):
        self.currentState = state

class Quit:
    def __init__(self, display, gameStateManager, user):
        self.display = display
        self.gameStateManager = gameStateManager
        self.font = pygame.font.SysFont('Lucida Console', 70)
        self.color_font = (200, 245, 10, 1)
        self.color_background = (65, 0, 168, 0.9)
        self.final_price = 0
        self.total_time = 0
        self.user = user

    def handle_quit(self):
        self.gameStateManager.set_state('quit')

    def precio_final(self):
        self.display.fill(self.color_background)
        price_text = self.font.render(f'Precio final: {round(self.final_price,
2)}€', True, self.color_font)
        minutos = int(self.total_time // 60)
        segundos = int(self.total_time % 60)
```

```
        time_text = self.font.render(f'Tiempo total de carrera:
{minutos}m:{segundos}s', True, self.color_font)
        price_text_rect = price_text.get_rect(center = (800, 350))
        time_text_rect = time_text.get_rect(center = (800, 450))
        self.display.blit(price_text, price_text_rect)
        self.display.blit(time_text, time_text_rect)
        pygame.display.update()
        pygame.time.wait(3000)

    def run(self):
        self.final_price =
self.gameStateManager.get_states()['taximetro'].get_score()
        self.total_time =
self.gameStateManager.get_states()['taximetro'].get_total_time()
        self.tiempo_minutos = self.total_time // 60
        self.tiempo_segundos = self.total_time % 60
        self.precio_final()
        self.today = date.today()
        self.d1 = self.today.strftime("%d/%m/%Y")
        datos_usuarios = pd.read_csv('Carreras.csv')
        df = pd.DataFrame({'Usuario' : [self.user], 'Fecha': [self.d1],
'Tiempo_Minutos' : [self.tiempo_minutos], 'Tiempo_Segundos':
[self.tiempo_segundos],'Precio': [round(self.final_price, 2)]})
        datos_usuarios = pd.concat([datos_usuarios, df], ignore_index = True)
        datos_usuarios.to_csv('Carreras.csv', index = False)
        pygame.quit()
        exit()

def init_game(user):
    game = Game(user)
    game.run()
```

Archivo: LogIn Screen.py

```
import pygame
from sys import exit

pygame.init()

#Tamaño de pantalla del juego
screen = pygame.display.set_mode((800, 400))
#Texto que figura en la ventana ejecutada
pygame.display.set_caption('Taximetro Interactivo')

#Reloj interno, para configurar la tasa de refreso del juego (fps)
clock = pygame.time.Clock()
```

```
first_screen = pygame.image.load('Graficos/road 2.jpg')
#Boton de LogIn
#Fuente para el boton de LogIn
login_font = pygame.font.SysFont('Lucida Console', 45)
#Rectangulo para el boton, (posicion ancho, posicion alto, tamaño ancho,
tamaño alto)
login_button_rect = pygame.Rect(585, 70, 170, 62)
#Superficie donde se coloca el boton
login_text = login_font.render('Log In', True, (128, 245, 10, 1))

#Boton de registo
reg_font = pygame.font.SysFont('Lucida Console', 45)
reg_but_rect = pygame.Rect(560, 170, 220, 62)
reg_text = reg_font.render('Registro', True, (128, 245, 10, 1))

#Boton de quit
quit_font = pygame.font.SysFont('Lucida Console', 45)
quit_but_rect = pygame.Rect(610, 270, 120, 62)
quit_text = quit_font.render('Quit', True, (128, 245, 10, 1))

#Bucle de ejecución del juego
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            if quit_but_rect.collidepoint(event.pos):
                pygame.quit()
                exit()

    a,b = pygame.mouse.get_pos()
    if login_button_rect.x <= a <= login_button_rect.x + 190 and
login_button_rect.y <= b <= login_button_rect.y + 70:
        pygame.draw.rect(screen, (91 ,23 ,202, 0.8), login_button_rect) #sin
hover
    else:
        pygame.draw.rect(screen, (65, 0, 168, 0.9), login_button_rect) #con
hover

    if reg_but_rect.x <= a <= reg_but_rect.x + 210 and reg_but_rect.y <= b <=
reg_but_rect.y + 70:
        pygame.draw.rect(screen, (91 ,23 ,202, 0.8), reg_but_rect)
    else:
        pygame.draw.rect(screen, (65, 0, 168, 0.9), reg_but_rect)

    if quit_but_rect.x <= a <= quit_but_rect.x + 210 and quit_but_rect.y <= b
<= quit_but_rect.y + 70:
```

```
        pygame.draw.rect(screen, (91 ,23 ,202, 0.8), quit_but_rect)
    else:
        pygame.draw.rect(screen, (65, 0, 168, 0.9), quit_but_rect)

    screen.blit(first_screen, (0, 0))
    screen.blit(login_text, (login_button_rect.x + 5, login_button_rect.y+5))
    screen.blit(reg_text, (reg_but_rect.x + 5, reg_but_rect.y+5))
    screen.blit(quit_text, (quit_but_rect.x + 5, quit_but_rect.y+5))

    pygame.display.update()
    clock.tick(60)
```

Archivo: Password database.py

```
import pandas as pd
import hashlib

#Cuando tengamos una base de datos como tal
#datos_usuarios = pd.read_csv("usuarios.csv")


def LogIn(cambio = False):
    '''
    LogIn va a permitir un argumento 'cambio' que por defecto será falso, para poder cambiar la contraseña.
    La función en primer lugar va a pedir un nombre al usuario y comprobará mediante condicional if si existe ese nombre en nuestra base de datos.
    Si el usuario no existe, entendemos que es un usuario nuevo y le pedimos que nos de un nombre
    La función comprueba que el nombre no esté ya en uso, y mientras no se de un nombre nuevo seguira pidiendolo en bucle.
    Después se le pide una contraseña "segura", podríamos definir que significa que su contraseña sea segura e incluso solicitar una longitud concreta (o carácteres)
    Una vez tenemos la contraseña, esta se hashea utilizando sha-256 y se almacena de forma encriptada en nuestra base de datos, para aumentar la seguridad.
    Para permitir cambiar la contraseña, se introduce una pregunta y respuesta secreta definidas por el usuario, y que deberá responder correctamente en el caso de necesitar cambiarla. Se permiten 3 intentos para acertar la respuesta.
    Finalmente el usuario, su contraseña encriptada y la pregunta y respuesta secretas se añaden a la base de datos de usuarios.
    Si el usuario si existe, se le pedirá que introduzca su contraseña y está se buscara en la base de datos en su forma encriptada.
    Si hay coincidencias el programa de taximetro se inicia, si no, se le pide que repita la contraseña, hasta un máximo de 6 intentos.
```

```
    '''

    datos_usuarios = pd.read_csv("Usuarios.csv")
    if cambio:
        usuario = input("Escriba su nombre de usuario: ")
        if usuario not in datos_usuarios["Usuarios"].values:
            print("Este usuario no existe")
        else:
            secret_q = datos_usuarios.query('Usuarios == @usuario')["Pregunta
Secreta"][1]
            secret_answ = hashlib.sha256(input(secret_q).encode('utf-
8')).hexdigest()
            true_answ = datos_usuarios.query('Usuarios ==
@usuario')["Respuesta Secreta"][1]
            intentos = 0
            if secret_answ == true_answ:
                new_pass = input("Introduce tu nueva contraseña: ")
                new_pass_hash = hashlib.sha256(new_pass.encode('utf-
8')).hexdigest()
                datos_usuarios.query('Usuarios == @usuario')["Pregunta
Secreta"][1]
                datos_usuarios.loc[datos_usuarios["Usuarios"] == usuario,
"Passwords"] = new_pass_hash
                datos_usuarios.to_csv('Usuarios.csv', index = False)
                print("Contraseña Cambiada")
            else:
                while (secret_answ != true_answ):
                    if intentos == 4:
                        print("Demasiados intentos fallados, reinicie el
programa")
                        break
                    else:
                        print("Respuesta incorrecta")
                        secret_answ = input(secret_q)
                        intentos += 1

    else:
        usuario = input("Escriba su nombre de usuario: ")
        if usuario not in datos_usuarios["Usuarios"].values:
            print("Usuario no registrado.")
            user = input("Escriba un nombre de usuario")
            while user in datos_usuarios.Usuarios.isin([user]):
                user = input("Ese nombre de usuario ya está seleccionado.
Escriba un nombre de usuario.")
            password = input("Escribe una contraseña segura")
            password_hash = hashlib.sha256(password.encode('utf-
8')).hexdigest()
```

```
            secret_q = input("Escriba una pregunta que solo usted conozca la
respuesta")
            secret_a = hashlib.sha256(input("Escriba la respuesta a su
pregunta secreta").encode('utf-8')).hexdigest()
            df = pd.DataFrame({'Usuarios': [user], 'Passwords':
[password_hash], 'Pregunta Secreta': secret_q, 'Respuesta Secreta': secret_a})
            datos_usuarios = pd.concat([datos_usuarios, df], ignore_index=
True)
            datos_usuarios.to_csv('Usuarios.csv', index = False)
        else:
            password_inp = input("Escriba su contraseña: ")
            password_inp_hash = hashlib.sha256(password_inp.encode('utf-
8')).hexdigest()
            intentos = 0
            password_local = datos_usuarios.query('Usuarios ==
@usuario')["Passwords"][1]
            while(password_inp_hash != password_local):
                if intentos == 6:
                    print("Demasiados intentos fallados, reinicie el
programa")
                    break
                else:
                    print("Contraseña incorrecta")
                    password_inp = input("Escriba su contraseña: ")
                    password_inp_hash =
hashlib.sha256(password_inp.encode('utf-8')).hexdigest()
                    intentos += 1

            else:
                print("Bienvenido")
                #taximetro()


def Iniciar(cambio = False):
    LogIn(cambio)

#Iniciar()

#Iniciar(True) Cambio de contraseña
```

Archivo: Taxi.ipynb

```
import time

class Taximetro:
    def __init__(self):
```

```
        self.costo_total = 0.0
        self.en_carrera = False
        self.en_movimiento = False
        self.ultimo_tiempo = None
        self.costo_por_segundo_parado = 0.02
        self.costo_por_segundo_movimiento = 0.05

    def iniciar_carrera(self):
        self.en_carrera = True
        self.en_movimiento = False
        self.ultimo_tiempo = time.time()
        print("La carrera ha comenzado. El taxi está parado.")

    def terminar_carrera(self):
        self.actualizar_costo()
        self.en_carrera = False
        print(f"La carrera ha terminado. El total es {self.costo_total:.2f}
Euros.")
        self.reiniciar()

    def comenzar_movimiento(self):
        if self.en_carrera and not self.en_movimiento:
            self.actualizar_costo()
            self.en_movimiento = True
            self.ultimo_tiempo = time.time()
            print("El taxi está en movimiento.")

    def detener_movimiento(self):
        if self.en_carrera and self.en_movimiento:
            self.actualizar_costo()
            self.en_movimiento = False
            self.ultimo_tiempo = time.time()
            print("El taxi está parado.")

    def actualizar_costo(self):
        if self.en_carrera:
            tiempo_actual = time.time()
            tiempo_transcurrido = tiempo_actual - self.ultimo_tiempo
            if self.en_movimiento:
                self.costo_total += tiempo_transcurrido *
self.costo_por_segundo_movimiento
            else:
                self.costo_total += tiempo_transcurrido *
self.costo_por_segundo_parado
            self.ultimo_tiempo = tiempo_actual

    def reiniciar(self):
        self.costo_total = 0.0
```

```
        self.en_carrera = False
        self.en_movimiento = False
        self.ultimo_tiempo = None
```