Práctica Python

DAW-DUAL

Práctica Programación
1r trimestre 2025/2026

Índice

¿Pero porque estoy aquí? Me quiero ir pa mi casa!

Primer día de curro. Te han dado el portátil, las credenciales, te han presentado al equipo y todo ha sido genial.

Segundo día de curro. Te llama tu responsable y te dice:

- Mira, para empezar, me gustaría que echaras un vistaciño a un proyecto que tenemos a medias. Lo estaba desarrollando un programador junior, pero se nos marchó la semana pasada. Creo que tiene funcionalidades ya implementadas, pero hay otras que faltan. Mírate la DOC que creo que hay hasta un UML. Cuando tengas algo me dices.

Vas todo ilusionad@ a revisar el proyecto y…. baixón “NO TE ENTERAS DE NADA!”

Tranquilidad, no cunda el pánico. ¡Lo sacarás adelante, estoy seguro!

Vamos a revisar esta vez juntos el código. (La siguiente vez seguramente lo harás tu sol@ ☹)

Revisando el proyecto

Buen percal cuando ves un proyecto que no es tuyo y aún por encima algo desordenado. Lo primero que debes de hacer es tirar del hilo. ¿A que me refiero? Busca el inicio de la ejecución (probablemente haya un “main”) y empieza a seguir la ejecución de la aplicación. La idea es que te hagas un esquema mental del flujo de la app, a qué métodos llama y porqué.

Revisa los archivos y verás que existe un main:

180 líneas ☹ … podría ser peor. En casi todos los proyectos de Python existe un _ _ main _ _ que determina la ejecución de la app.

```
180 ▷ if __name__ == "__main__":
181         while not login():
182             pass
183         menu()
```

Parece que mientras el Login no sea exitoso no se despliega el menú principal.

Antes de ejecutar la app. Tenemos un requirements.txt. Ábrelo. Solo tenemos una dependencia ☺

Tienes 2 opciones.

1. Hacer directamente un "pip install pygame"
2. O "pip install -r requirements.txt"

El segundo método está mejor sobre todo cuando el proyecto tiene muchas dependencias. Básicamente pip se encarga de leer en archivo e instalarlas.

Una vez realizado esto, ejecuta la app y pruébala.

Trae un usuario creado por defecto:

- Username = sergio
- Password = abc123.

Vamos a ver ahora que hace la función login(). (Mantén pulsado crtl y haz click en la función y te llevará directamente a ella)

```
86      def login():  1 usage
87          from login import Login
88          print_logo()
89          global current_user
90          login = Login()
91          login.load_users()
92
93          while True:
94              print("\t\t\t\t1. Login\t\t\t\t2. Create User")
95              print("\t\t\t\t\t\t     0. Salir")
96              print("\n*********************************************************************")
97
98              choice = input("Elige una opción: ")
99
100             if choice == '1':
101                 if login.is_login_correct():
102                     current_user = login.get_current_user()
103                     return True
104             elif choice == '2':
105                 login.create_new_user()
106             elif choice == '3':
107                 print("Saliendo de la aplicación.")
108                 return False
109             else:
110                 print("Opción no válida. Inténtalo de nuevo.")
```

Vale, de aquí se pueden sacar cositas.

1. Hay un módulo Login que tiene pinta de gestionar los logins (bien, al menos parece que el desarrollador está siendo coherente con el principio Single Responsability)
2. Hay una función print_logo() que entiendo que hace lo que dice ☺
3. Pero lo importante viene ahora:
    a. Hay una variable global llamada current_user en el main (que si te fijas, si el login en positivo le da el valor del usuario – línea 102)
    b. Crea una new Class en la variable login (línea 90)
    c. Además, antes carga los usuarios del sistema (línea 91)
4. Por último, aquí la aplicación se bifurca. Por un lado, hay una lógica de autenticación de usuario y por otra la de crear usuario. Evidentemente la de crear usuario debería de tener un "hilo" más corto. Vamos a ver qué hace.

Mantén pulsado crtl y haz click en create_new_user() – línea 105

```
42      def create_new_user(self):  1 usage
43          """Permite crear un nuevo usuario y guarda su hash en el fichero."""
44          print("\n--- CREAR NUEVO USUARIO ---")
45
46          username = input("Elige nombre de usuario: ")
47
48          # Comprobar si ya existe
49          for user in users:
50              if user.username == username:
51                  print(f"Error: El usuario '{username}' ya existe. Intenta iniciar sesión.")
52                  return
53
54          password = input("Elige contraseña: ")
55          hashed_pwd = self.hash_password(password)
56
57          # 1. Guardar en el archivo (añadir modo 'a' para append)
58          self.create_login_directory()
59          try:
60              with open(USERS_FILE_PATH, 'a', encoding='utf-8') as f:
61                  f.write(f"{username},{hashed_pwd}\n")
62
63              # 2. Recargar la lista global 'users'
64              self.load_users()
65
66              print(f"\n¡Usuario '{username}' creado y registrado exitosamente!")
67          except Exception as e:
68              print(f"Error al guardar el usuario: {e}")
```

Vamos a ver este código:

Leyendo un poco en diagonal, parece que:

1. Login matiene una lista de usuarios del sistema en la lista llamada users. Fíjate en la línea 49 que recorre los usuarios por si el nombre que propone el nuevo usuario ya existiera.
2. En la línea 55 está hasheando la password (Genial, al menos el junior no la mete en raw)
3. Las siguientes líneas si te fijas, todo huele a que se está guardando en un fichero y me parece que la constante USERS_FILE_PATH tiene mucho que ver.

4. Si mantiene pulsado crtl y haces click en la variable te lleva a donde está declarada.
5. Vale, lo tienes: Parece que los usuarios se están guardando en: "./login/users.txt"
6. Vamos a ver el archivo, a ver que contiene.

```
7. sergio,f2e53c927c66fe711e8e88ef9b37a8e3187f1652216b313fc8eb2513883dd360
```

8. usuario, hash. OK

Ahora mismo te puedes hacer una idea global de cómo se gestionan los usuarios en el sistema de login. Cuando se crea, se escribe en el archivo y cuando se loguea (aunque aún no lo hemos verificado) la lógica nos dice de que leerá del archivo y verificará si el user y la pass coinciden.

Vamos a volver a main.py. El hilo de create_new_user ya lo hemos visto. Vamos ahora con el hilo principal que sería el login.

Línea clave 101:

```
if login.is_login_correct():
```

Lo de siempre: ctrl + pincha en la función.

```
70      def is_login_correct(self):  1 usage
71          """Función para autenticar un usuario existente."""
72          global current_user
73
74          # Aseguramos que la lista 'users' esté poblada antes de buscar
75          if not users:
76              print("No hay usuarios registrados para iniciar sesión.")
77              return False
78
79          username = input("Ingresa tu nombre de usuario: ")
80          password = input("Ingresa tu contraseña: ")
81
82          # Hasheamos la contraseña ingresada para compararla con el hash guardado
83          input_hashed_pwd = self.hash_password(password)
84
85          for user in users:
86              # Comparamos el nombre y el HASH de la contraseña
87              if user.username == username and user.password == input_hashed_pwd:
88                  print(f"\n¡Bienvenido de nuevo, {username}!")
89                  current_user = user
90                  current_user.charge_playlists_from_files()
91                  return True
92
93          # Si el bucle termina sin un match:
94          print("\nUsuario o contraseña incorrectos.")
95          return False
```

Leemos en diagonal:

1. tiene una variable global current_user (como en el main. Parece que cuando el login es exitoso le setea un valor)
2. Si no hay usuarios parece que sale del login
3. Pregunta por usuario y password y hashea la pass.
4. Y ahora parece que en la línea 85 empieza el bucle para verificar que el usuario y la password existe.
5. Si existe, current_user es igual a user encontrado (línea 89)
6. Y en la siguiente línea, parece que llama a una función para cargar las playlist del usuario de un archivo. WOW! Parece importante esta llamada. Vamos a cheirar.
7. Ctrl + click

```
24      def charge_playlists_from_files(self):  1 usage (1 dynamic)
25          import os
26          import main
27          from playlist import PlayList
28
29          user_directory = f"./{self.username}/"
30          if not os.path.exists(user_directory):
31              return
32
33          for filename in os.listdir(user_directory):
34              if filename.endswith(".txt"):
35                  playlist_name = filename[:-4]
36                  playlist = PlayList(playlist_name, self)
37
38                  with open(os.path.join(user_directory, filename), 'r', encoding='utf-8') as f:
39                      for line in f:
40                          song_name = line.strip()
41                          if song_name:
42                              for song in main.songs:
43                                  if song.name == song_name:
44                                      playlist.add_song(song)
45
46                  self.playlists.append(playlist)
```

Vamos a intentar entender este método:

1. Vemos que por los imports va a utilizar la clase PlayList
2. Parece que tiene sentido que el propio usuario cree sus playlist. Os acordáis del ejemplo de mano y dedo. No tiene sentido que exista el dedo sin la mano, pues podemos aplicar eso mismo a que no tiene sentido que exista una playList creada por un usuario sin que el usuario exista. Con lo cual, la responsabilidad de crear el objeto PlayList es de User. (Parece que el programador junior sabe cosas ☺)
3. Línea 29: Parece que va a usar un directorio con el nombre de usuario (vete a ver al proyecto a ver si hay una carpeta con el nombre de “sergio” que es el usuario que de momento está creado)
4. A partir de la línea 33 lo que hace es: recorre los ficheros y si acaban en .txt crea una playList con el nombre del fichero. Línea 38, abre el fichero, y por cada línea compara si existe esa

canción en la línea de main.songs (todas las canciones del sistema). Si es así, entonces añade la canción a la playList.

OK. Resumiendo. Las playlist de los usuarios se guardan en ficheros.txt dentro de sus carpetas. Y además, coge las canciones del main.songs. Así que vamos a ver entonces donde están las canciones.

Vete a main y busca “songs” seguramente como variable al principio.

```
11    songs = [Song( name: "circles",  duration: "3:35",  release_date: "2019-08-30", Genre.POP),
12             Song( name: "birds_of_a_feather",  duration: "4:14",  release_date: "2021-11-19", Genre.POP)]
```

Vale, no es muy elegante, pero la mete a pecho. ¿Pero me pregunto ahora? ¿Y cómo suena? Donde están los archivos .mp3? Vamos a seguir el flujo de la app hasta ver como reproduce una canción.

```
180  if __name__ == "__main__":
181      while not login():
182          pass
183      menu()
```

Después del login exitoso se ejecuta la función menu()

```
113  def menu():  1 usage
114      while True:
115          print_logo()
116
117          print("\t1. Reproducir PlayList\t\t\t2. Reproducir Canción")
118          print("\t3. Crear Playlist\t\t\t\t4. Tu Wrapped")
119          print("\t\t\t\t\t\t  0. Salir")
120          print("\n*********************************************************************")
121
122          try:
123              choice = int(input("Elige una opción: "))
124
125              if choice == 1:
126                  for playlist in current_user.playlists:
127                      print(f"- {playlist.name}")
128                  reproduce_playList(input("Ingresa el nombre de la playList: "))
129
130              elif choice == 2:
131                  for song in songs:
132                      print(f"- {song.name}")
133                  reproduce_song(input("Ingresa el nombre de la canción: "))
134
```

Nos interesa la opción 2, para ver realmente como ejecuta canciones.

Primero podemos ver que en la línea 131 recorre las canciones almacenadas en Spotify y las muestra para que el usuario la vea.

Línea 133 llama a la función reproduce_song y le pasa como parámetro la canción.

Vamos a ir a reproduce_song a ver qué hace.

```
15  def reproduce_song(song):  1 usage
16      # Inicializar el mixer
17      pygame.mixer.init()
18
19      # Cargar canción
20      pygame.mixer.music.load(MEDIA_DIR + song + ".mp3")
21
22      # Reproducir
23      pygame.mixer.music.play()
24
25      print(f"Reproduciendo {song}... ")
26      while True:
27          comando = input("Pulsa (pausa, reanudar, stop, salir): ").strip().lower()
28
29          if comando == "pausa":
30              pygame.mixer.music.pause()
31          elif comando == "reanudar":
32              pygame.mixer.music.unpause()
33          elif comando == "stop":
34              pygame.mixer.music.stop()
35          elif comando == "salir":
36              pygame.mixer.music.stop()
37              break
38          else:
39              print("Comando no válido")
```

Vale, parece que utiliza la librería de pygame para reproducir .mp3.

Por otro lado, fíjate en la línea 20 (Es la clave). Parece que carga la canción desde MEDIA_DIR + el nombre de la canción + .mp3. así que vamos a ver arriba en las constantes que es MEDIA_DIR

```
MEDIA_DIR = "./media/"
```

Vete a ver que hay en /media/ del proyecto.

Debería de haber 2 canciones. Perfecto, así que cuando se da de alta una canción, ejemplo:

```
Song("birds_of_a_feather", "4:14", "2021-11-19", Genre.POP)
```

Además, necesitamos meter en /media/ el mp3 con el mismo nombre.

Bueno, creo que lo más importante ya está revisado. Te dejo a ti que sigas investigando algo más de código.

Qué debes hacer

1. Primero (que sé que tienes ganas), créate un usuario y móntate unas PlayList con tus .mp3. Créate al menos 2 playlist y además añádele bien de canciones ☺
2. Reprodúcelas y cacharrea a ver si todo va bien.

3. Crear la clase Artist como se especifica en el UML. De momento no hace falta que crees Album. Es muy importante cuando crees la clase Artist mínimo vayas al constructor de Song y añadas que se le pase un artista. Esto no hará falta a posteriori para sacar estadísticas del wrapped.

4. Falta por implementar el case 4. “Tu Wrapped”. ¿Qué debe de mostrar?

    a. Tu nombre de usuario.
    b. Cuantas playList tienes
    c. Tus nombres de las playList y cuantas canciones tienen cada una.
    d. Tu canción más repetida (se entiende que la que más aparezca en las playList)
    e. Tu genero favorito
    f. Tu artista favorito