Análisis del Proyecto

Este proyecto se basa en la creación de un casino online llamado “DreamWorld Casino”, que consta de un menú principal con diferentes funcionalidades, como el registro de un usuario, el cual toma información del usuario y posteriormente la guarda, el DreamWorld Casino, en el cual ya se puede jugar directamente, también el programa busca tener una configuración avanzada, donde se puedan cambiar datos internos directamente, como lo puede ser un usuario o el monto mínimo de las fichas

Tabla de contenido

Avance 1

Se crean tres archivos por aparte donde se almacenan distintas partes del programa:

- **menu**, donde se imprimen las opciones del menú principal y el usuario elige una de las opciones.
- **user_registration**, donde la persona crea su usuario
- **index**, donde según la opcion elegida por el usuario sigue uno de los procesos

Menú Principal

Menu

1. Se crea el método (**def**) **printMainMenu,** donde se imprimen las distintas opciones del menú principal
2. Se crea el método (**def**) **getMenuOption**
    a. Crear variable **menuOption**, donde se le pregunta al usuario cuál opción del menú
    b. Se devuelve **menuOption**
3. Se crea el método (**def**) **isValidMenuOption,** que espera el valor **menuOption**
    a. Se devuelve un boolean “true” si **menuOption** es mayor igual a 1 y menor igual a 4
4. Se crea el método (**def**) **validateMenuOption,** que espera un valor **menuOption**
    a. Se usa un if, que valida si **no** se cumple el método **isValidMenuOption**
        i. Se imprime un mensaje de error
        ii. Se vuelve a correr el menú principal desde el inicio
    b. Si se cumple el método devuelve **menuOption**
5. Se crea el método (**def**) **runMainMenu,** donde se llama a todos los métodos anteriores.
    a. Se llama el método antes creado **printMainMenu**
    b. Se crea la variable **menuOption,** donde se guarda el valor que devuelve **getMenuOption**
    c. Se devuelve la acción o valor de **validateMenuOption,** llevando la variable creada **menuOption**

Registro de Usuario

User_Registration

1. Se importa la biblioteca **getpass**
2. Se crea la variable **userIdAttempts**, donde se van a ir guardando el número de intentos de ingreso de ID
3. Se crea el método (**def**) **validateUserIdAttempts,** donde se van a validar el número de intentos
    a. Se crea la variable **totalUserIdValidAttempts** y se le asigna 3
    b. Se hace global la variable **userIdAttempts,** para que se guarde por fuera del método. ("**global**" se utiliza para declarar una variable dentro de una función como una variable global, lo que significa que la variable se puede utilizar tanto dentro de la función como fuera de ella, en todo el ámbito del programa.
    c. Para el caso de nuestro proyecto, estas son las justificaciones de su uso:
    d. - Compartir información entre múltiples funciones
    e. - Configuración global
    f. - Mantener un estado global)
    g. Se le suma 1 a **userIdAttempts,** cuando se invoque a este método
    h. Se usa un **if**, para validar si **userIdAttempts** es igual a **totalUserIdValidAttempts**

        i. Se imprime un mensaje de que ha excedido los intentos de ingreso
        ii. Se importa el método **start** del archivo **index**
        iii. Se invoca el método **start**
    i. Si no se cumple, se llama al método **addRegistration**, el cual es creado al final de este archivo (**user_registration**)
4. Se crea el método (**def**) **isValidUserId,** donde se va a validar el mínimo del ID de usuario, que espera el valor **userId**
    a. Se crea la variable **minUserIdLenght**, el cual debe ser 5
    b. Devuelve un valor booleano, si el tamaño(**len**) de **userId** es mayor o igual a **minUserLenght**
5. Se crea el método (**def**) **validateUserId,** que espera el valor **userId**
    a. Se usa un **if**, para validar el booleano que devuelve **isValidUserId,** llevando el valor **userId**
        i. Si se cumple el mínimo de caracteres y por lo tanto, el método, devuelve el ID **userId**
        ii. Si no devuelve un mensaje de error de "ID de usuario invalido"
        iii. Se llama al método **validateUserIdAttempts**, antes creado para ir midiendo la cantidad de intentos de ingreso de ID
6. Se crea el método (**def**) **getUserId,** donde como tal se le pide al usuario que ingrese su ID.
    a. Se crea la variable **userId,** para guardar el ingreso del ID
    b. Devuelve el resultado del método **validateUserId**, llevando el ingreso del usuario (**userId**)
7. Se crea el método (**def**) **createPIN,** para solicitar al usuario que cree su PIN
    a. Se usa un **while**, para que se repita hasta que el usuario ingrese un PIN correcto y lo devuelva.
        i. Se usa un **try,** para solicitar el PIN del usuario
            1. Se crea la variable **userPin** que guarda el ingreso del PIN con la biblioteca **getpass** y lo convierte en **string**
            2. Se usa un **if** para validar que el tamaño (**len**) del PIN sea mayor o igual a 6.
                a. Devuelve el PIN ingresado **userPin**
            3. Con un **else,** si no se cumple se imprime un mensaje de que el PIN debe ser de al menos 6 dígitos
        ii. Si el PIN que ingresa tiene algo más que números, entra en el **except**.
            1. Muestra un mensaje de "ingresar solo números"
8. Se crea el método (**def**) **authenticatePin,** para confirmar el PIN creado por el usuario y espera el valor **userPin.**
    a. Se usa un **while**, para que se repita hasta que el usuario ingrese el mismo PIN creado anteriormente.
        i. Se usa un **try,** para solicitar la confirmación del PIN
            1. Se crea la variable **confirmPin** que guarda el ingreso del PIN con la biblioteca **getpass** y lo convierte en **int**
            2. Se usa un **if** para validar que el PIN creado (**userPin**) sea igual al PIN de confirmación.
                a. Se imprime un mensaje de confirmación de que se creó el PIN con éxito.
                b. Devuelve el PIN ingresado **userPin**

3. Con un **else,** si no se cumple se imprime un mensaje de que el PIN no coincide con el creado anteriormente

ii. Si el PIN que ingresa tiene caracteres no númericos, entra en el **except**.

1. Muestra un mensaje de "ingresar solo números"

9. Se crea el método (**def**) **getUserPin,** donde se obtiene el PIN ingresado por el usuario.

- **a.** Se crea la variable **userPin** que se le asigna lo que devuelve el método **createPIN**
- **b.** Se invoca el método **authenticatePin,** donde se envía el valor de **userPin**
- **c.** Se devuelve el PIN del usuario **userPin**

10. Se crea el método (**def**) **getDeposit,** donde se va a depositar el dinero inicial mínimo.

- **a.** Se crea la variable global **minMoney**
- **b.** A la variable **minMoney**¸ se le asigna 10000 (como mínimo temporal)
- **c.** Se crea la variable **depositAttempts,** para el numero de intentos y se le asigna 0.
- **d.** Se crea la variable **depositMoney,** para el ingreso de dinero del usuario
- **e.** Se utiliza un **while** que se repite mientras el número de intentos sea diferente o igual a 3 **y** el dinero depositado sea menor al mínimo de dinero.

i. Se usa un **try** para empezar con el ingreso de dinero

1. En la variable **depositMoney,** se guarda el ingreso de dinero del usuario y lo convierte a **float**
2. Se usa un **if** para validar que el dinero depositado sea mayor o igual al mínimo
   - **a.** Se imprime un mensaje de confirmación
   - **b.** Se importa el método **start** del archivo **index**
   - **c.** Invoca al método **start**, para que se devuelva al menú principal
3. Con un **else**, si no se cumple
   - **a.** Se crea la variable **totalDepositValidAttempts** y se le asigna 3
   - **b.** A la variable antes creada **depositAttempts** se le suma 1
   - **c.** Se usa un **if,** donde se valida si el número de intentos hechos por el usuario (**depositAttempts**), es igual al total permitido (**totalDepositValidAttempts**)
     - **i.** Imprime que se ha excedido los intentos
   - **d.** Si no se cumple:
     - **i.** Se crea la variable de **attemptsLeft,** que se le asigna la resta de **totalDepositValidAttempts y depositAttempts**
     - **ii.** Se imprime que el monto no es válido y los intentos restantes que le quedan al usuario (**attemptsLeft**)

ii. Entra en el **except,** si el usuario ingresa caracteres no numéricos

11. Se crea el método (**def**) **addRegistration,** donde se obtienen los datos de registro del usuario

- **a.** Se imprime un mensaje de Registro de Usuario
- **b.** Se crea una variable global **userId**
- **c.** Se crea una variable global **userName**
- **d.** Se crea una variable global **userPin**
- **e.** A **userId** se le asigna lo que devuelve el método **getUserId**

f. A **userName** se le asigna el ingreso del usuario del nombre
g. A **userPin** se le asigna lo que devuelve el método **getUserPin**
h. Se llama al método **getDeposit**

Index

Index

1. Se importa el método **runMainMenu** del archivo **menú**
2. Se importa el método **addRegistration** del archivo **user_registration**
3. Se crea el método (**def**) **start**, para correr el menú principal
    a. Se crea la variable **menuOption** y se le asigna el método **runMainMenu**
    b. Se usa un **if** para validar si la opción elegida en el menú principal es 1:
        i. Se llama **addRegistration**
    c. Se usa un **elif** validar si la opción elegida en el menú es 2:
        i. Se imprime un mensaje de que la opción no está disponible (temporal hasta que esté terminada la sección)
        ii. Se devuelve al menú principal
    d. Se usa otro **elif** validar si la opción elegida en el menú es 3:
        i. Se imprime un mensaje de que la opción no está disponible (temporal hasta que esté terminada la sección)
        ii. Se devuelve al menú principal
    e. Se usa otro **elif** validar si la opción elegida en el menú es 4:
        i. Se imprime un mensaje de “Volviendo al menú principal”
        ii. Se devuelve al menú principal
4. Se invoca al método **start** para que empiece a correr el algoritmo.

Estructuras usadas:

- if, elif, else
- for
- while

Otros:

- try except
- def
- print
- input
- getpass
- len()
- int()
- float()
- str()

Avance 2

Registro de Usuario

Función getUserInfo()

Retorna un Array con la información del usuario, el objetivo es englobar funcionalidades que se considera están bajo un mismo nivel de abstracción, debido a que recopila los datos del usuario. El orden de los datos retornados en el Array es el siguiente: [userId, userName, userPin]

Función getUsername()

Se encarga de solicitar y retornar el nombre al usuario, en formato de cadena de texto.

Función startUserRegistration()

Función de entrada al módulo, engloba los pilares para el registro de un usuario, compuesto por las funciones que obtienen la información del usuario, obtiene el depósito y añade finalmente el registro en la estructura de archivos, finalmente retorna al menú principal.

Función isUserExist(userId)

Función booleana, recibe por parámetro un Id, verifica si existe o no registro de alguna carpeta con el registro de este ID, de no ser así retorna falso, esto se logra gracias a la librería ‘os’ con su método exists, el cuál recibe la ruta del archivo por parámetro.

Función addRegistration()

Función que engloba otros métodos requeridos para el registro del usuario, como el guardado de la información del depósito y el guardado de información de sus credenciales.

Función setDepositMoney()

Obtiene la información del usuario, como el id, username, pin, posteriormente con la librería ‘os’ se crea el directorio con el identificador del usuario dentro de la carpeta ‘users/’ , para luego generar un archivo llamado ‘saldos.txt’ dentro y guardar ahí el depósito realizado por el usuario, ésto mediante la estructura ‘with’ la cual permite cerrar el archivo abierto pase lo que pase, y también se utiliza el método open() que recibe como primer parámetro la ruta del archivo y como segundo el modo de escritura.

Función setCredentials()

Obtiene la información del usuario, y abre mediante open() el archivo ‘usuarios_pines.txt’, este archivo es el encargado de almacenar las credenciales del usuario, siendo estas el ID y el Pin, al igual que la función setDepositMoney() se utiliza with para cerrar el archivo después de ser usado y write() para escribir dentro del archivo.

Estructuras nuevas

Estructura ‘with’, utilizada para cerrar los archivos abiertos después de su uso, pase lo que pase.

Librerías nuevas

Librería ‘os’, utilizada para crear carpetas, verificar si un archivo existe, en general permite hacer llamadas del sistema operativo.

PIN

Función createPin()

Esta función solicita al usuario que cree su PIN. El PIN debe contener al menos 6 dígitos. Si el usuario ingresa un valor con menos de 6 dígitos, se le pedirá que lo intente nuevamente.
La función entra en un bucle **while True** para asegurarse de que el usuario proporcione un valor válido. Utiliza **getpass.getpass()** para ocultar la entrada del PIN al usuario.
Si el PIN ingresado es válido (tiene al menos 6 dígitos), la función devuelve el PIN como string.

Función authenticatePin(userPin)

Esta función autentica el PIN del usuario pidiéndole que lo confirme ingresándolo nuevamente. Recibe el siguiente parámetro:

- **userPin**: El PIN ingresado previamente por el usuario como string.

La función entra en un bucle **while True** para asegurarse de que el usuario proporcione un valor válido. Al igual que en **createPIN()**, utiliza **getpass.getpass()** para ocultar la entrada del PIN al usuario.
Si el PIN ingresado coincide con el PIN previamente ingresado (**userPin**), se muestra un mensaje de éxito y se devuelve el **userPin**.

Función getUserPin()

Esta función es la principal y maneja el proceso completo de creación y autenticación del PIN del usuario. No recibe ningún parámetro.
La función llama a **createPIN()** para que el usuario cree un PIN. Luego, llama a **authenticatePin(userPin)** para autenticar el PIN ingresado.
Si el PIN es autenticado con éxito, se muestra un mensaje de éxito y se devuelve el **userPin**.

Depósito

Método printMenuMoneyType()

Esta función simplemente imprime un menú con tres opciones de depósito: Colones, Dólares y Bitcoin.

Función attemptsDeposit(depositMoney, depositAttempts, type)

Esta función gestiona los intentos de depósito y verifica si el monto ingresado es válido. Recibe los siguientes parámetros:

- **depositMoney**: El monto a depositar.
- **depositAttempts**: El número de intentos de depósito realizados.
- **type**: El tipo de moneda seleccionada para el depósito (1 para Colones, 2 para Dólares, 3 para Bitcoin).

La función primero determina el valor mínimo de depósito según el tipo de moneda seleccionado. Luego, verifica si el monto ingresado es mayor o igual al valor mínimo. Si es así, se considera el depósito como realizado con éxito y se devuelve **depositMoney**, **3** (número máximo de intentos permitidos) y **True** para **flagDeposit**.
Si el monto no alcanza el valor mínimo, la función aumenta el número de intentos **depositAttempts** en 1. Si el número de intentos alcanza el límite máximo (**totalDepositValidAttempts** es igual a 3), se muestra un mensaje y se regresa al menú principal.
Si todavía hay intentos disponibles, se muestra un mensaje indicando el monto mínimo requerido y cuántos intentos quedan.

convertMoney(depositMoney, moneyType): Esta función convierte el monto ingresado a una moneda específica, según el tipo de moneda seleccionado. Recibe los siguientes parámetros:

- **depositMoney**: El monto a depositar.
- **moneyType**: El tipo de moneda seleccionado para el depósito (1 para Colones, 2 para Dólares, 3 para Bitcoin).

La función obtiene los valores de cambio para el Dólar a Colones y el Dólar a Bitcoin a través de la función **helpers.tipoDeCambio()**. Luego, realiza la conversión del monto de acuerdo con la moneda seleccionada.

Función getDeposit(depositMoney, depositAttempts)

Esta función es la principal y maneja el proceso de depósito. Recibe los siguientes parámetros:

- **depositMoney**: El monto a depositar.
- **depositAttempts**: El número de intentos de depósito realizados.

La función inicia un bucle que se repite hasta que el número de intentos sea igual a 3. En cada iteración, muestra el menú de opciones de depósito y solicita al usuario que ingrese el número correspondiente al tipo de moneda que desea depositar.
Según la opción seleccionada, se solicita al usuario que ingrese el monto a depositar. Luego, se convierte el monto a dólares utilizando la función **convertMoney**. Después, se llama a la función **attemptsDeposit** para verificar si el monto ingresado es válido.
Si el depósito se realiza con éxito (**flagDeposit** es **True**), se sale del bucle y se devuelve el **depositMoney** y **True**. Si no, se sigue solicitando al usuario que ingrese un monto válido hasta que se alcancen los tres intentos o se realice un depósito válido

Autenticación de Usuario

Se crean 2 archivos por aparte donde se almacenan 2 nuevas partes del programa:

- **helpers**, donde se van a ir añadiendo los métodos y funciones que usemos repetidamente durante el proyecto.
- **user_authentication**, donde la persona se logea e ingresa a su cuenta
- Se crea un nuevo file llamado “user_authentication”
    - Se importa el archivo “helpers”
    - Se importa la biblioteca “getpass”
    - Se importa el archivo “user_registration”
    - Se crea una variable global “userNameTxt”, para poder usarla y cambiarla en distintas funciones
    - Se crea el metodo “login” para hacer la autenticación del usuario y que pueda ingresar a su cuenta
        - Variable “userIdAttempts” inicializada en 0, para ir guardando los intentos de ingreso de ID
        - Variable “userPinAttempts” inicializada en 0, para ir guardando los intentos de ingreso de PIN
        - Variable “totalValidAttempts” inicializada en 3, para guardar el número total de intentos
        - Variable “userIdTxt” inicializada en “jai”, que guarda un ID temporal para validar el usuario. **En un futuro se va a guardar el ID traido del txt**
        - Variable “userNameTxt” inicializada en “Jairell”, que guarda un nombre temporal. **En un futuro se va a guardar el nombre del usuario traido del txt.**

- Variable “userMoneyTxt ” inicializada en “10000”, que guarda una cantidad de deposito temporal. **En un futuro se va a guardar el depósito traido del txt.**
- Variable “userPinTxt” inicializada en “123456”, que guarda un nombre temporal. **En un futuro se va a guardar el nombre del usuario traido del txt.**

- Se implementa un while que se repita mientras que los intentos de ingreso de ID sea diferente a 3
  - Se crea variable “userId” para pedirle al usuario que ingrese su ID
  - Se usa un if para validar que el “userId” ingresado, sea igual “userIDTxt” que vendra desde el txt en un futuro (por el momento debe ser igual a “jai”)
    - Se implementa otro while para que se repita mientras que los intentos de PIN sea diferente a 3
      - Se crea la variable “userPin” para que el usuario digite su PIN.
      - Se usa otro if para validar que el pin ingresado sea igual al del txt:
      - Si se cumple se llama al metodo “**menuCasino”**
      - Se devuelve el el userId, userPin, userNameTxt, userMoneyTxt, para guardarlos luego
      - Si no se cumple, else:
      - Se le suma y asigna 1 a “userPinAttempts”
      - Se usa otro if para validar la cantidad de intentos, si los intentos de PIN son iguales al total de intentos antes planteados (3)
        - Se imprime un mensaje que muestra que se han excedido los intentos de ingreso de PIN
        - Se llama a la funcion de helpers, “returnToMainMenu”, para volver al menu principal
      - Si no se cumple, else
        - Se crea la variable “attemptsLeft” para tener la cantidad entre la resta de 3 y el numero de intentos de PIN
        - Se imprime un mensaje de que el dato ingresado no es valido y el numero de intentos restantes (attemptsLeft)
  - Si no se cumple la igualdad entre los intentos de ID y el número total de intentos (3):
  - Se le suma y asigna 1 a los intentos del usuario de ID
  - Se implementa un if que valide la cantidad de intentos del usuario de ID y la cantidad total
    - Se imprime un mensaje de que ha excedido los intentos de ID
      - Se llama a la funcion de helpers, “returnToMainMenu”, para volver al menu principal

    - Si no se cumple, else
        - Se crea la variable “attemptsLeft” para tener la cantidad entre la resta de 3 y el numero de intentos de ID
        - Se imprime un mensaje de que el dato ingresado no es valido y el numero de intentos restantes (attemptsLeft)
  - Se crea el método “menuCasino”, donde se van a presentar las opciones del DreamWorld Casino
      - Se imprime un mensaje de bienvenida
      - Se implementa un while
          - Se imprime la primera opcion del menú (“Retirar Dinero”)
          - Se imprime la segunda opcion del menú (“Depositar Dinero”)
          - Se imprime la tercera opcion del menú (“Ver saldo actual”)
          - Se imprime la cuarta opcion del menú (“Juegos en línea”)
          - Se imprime la quinta opcion del menú (“Eliminar usuario”)

Estructuras usadas:

- if, elif, else
- for
- while
-

Otros:

- try except
- def
- print
- input
- getpass
- os
- len()
- int()
- float()
- str()
- global