Twitter

El objetivo es desarrollar una aplicación que *simule* (de manera simplificada) la red social **Twitter** desde línea de comandos usando el paradigma de *Programación Orientada a Objetos* con Python junto al módulo sqlite para el *acceso a datos*.

1. Funciones globales

Implementa, *al menos*, las siguientes **funciones** globales:

```
def create_db(db_path: str) -> None:
```

- Crea la base de datos en la ruta `db_path`
- Crea la tabla `user`:

`id`	Clave primaria (identificador numérico)
`username`	Nombre de usuario (**único**)
`password`	Contraseña de usuario
`bio`	Biografía de usuario

- Crea la tabla `tweet`:

`id`	Clave primaria (identificador numérico)
`content`	Contenido del *tweet*
`user_id`	**Clave ajena** al usuario que escribió este *tweet**
`retweet_from`	**Clave ajena** al *tweet* que se retuiteó (si es el caso)*

* No olvides las restricciones de clave ajena.

2. Clase `User`

Escribe una clase `User` que represente un usuario de *Twitter*.

2.1. Atributos de clase

Crea, *al menos*, los siguientes **atributos de clase**:

`con`

- Conexión a la base de datos `DB_PATH`.
- *Resultados de consulta* en **modo fila**.

`cur`

- Cursor creado desde la conexión a la base de datos.

2.2. Métodos

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, username: str, password: str, bio: str = '', user_id: int = 0):
```

- Crea los atributos *homónimos* a los parámetros.
- Crea el atributo `logged` de la forma correspondiente.

```
def save(self) -> None:
```

- Guarda en la base de datos el objeto `self`.
- Actualiza el atributo `id` de `self` a partir de lo que devuelve la sentencia de inserción.

```
def login(self, password: str) -> None:
```

- Realiza el *login* del usuario `self` con la contraseña `password`.
- Si la contraseña pertenece al usuario y es correcta, debes *actualizar* los atributos correspondientes de `self`.

```
def tweet(self, content: str) -> Tweet:
```

- Si el usuario `self` no está logeado debes lanzar una excepción de tipo `TwitterError` con el mensaje: `User <username> is not logged in!`
- Si el contenido supera los 280 caracteres debes lanzar una excepción de tipo `TwitterError` con el mensaje: `Tweet has more than 280 chars!`
- Construye (y devuelve) un objeto `Tweet` con el contenido indicado.
- Almacena dicho *tweet* en la base de datos.
- Utiliza la clase `Tweet` y *sus métodos*.

```
def retweet(self, tweet_id: int) -> Tweet:
```

- Si el usuario `self` no está logeado debes lanzar una excepción de tipo `TwitterError` con el mensaje: `User <username> is not logged in!`
- Si `tweet_id` no existe en la base de datos debes lanzar una excepción de tipo `TwitterError` con el mensaje: `Tweet with id <id> does not exist!`
- Construye (y devuelve) un objeto `Tweet` que sea un *retweet* del *tweet* indicado por `tweet_id`.
- Almacena dicho *tweet* en la base de datos.
- Utiliza la clase `Tweet` y *sus métodos*.

```
def tweets(self):
```

- Es una **propiedad**.
- Es una **función generadora**.

- Devuelve todos los tweets propios del usuario `self`.
- Devuelve objetos de tipo `Tweet`.
- Usa el método `from_db_row()` de la clase `Tweet`.

```
def __repr__(self):
```

- Devuelve la representación del usuario `self` con el formato: `'<username>: <bio>'`

```
def from_db_row(cls, row: sqlite3.Row) -> User:
```

- Es un **método de clase**.
- Construye (y devuelve) un objeto de tipo `User` a partir del argumento `row` que es una fila de consulta desde la base de datos.

3. Clase `Tweet`

Escribe una clase `Tweet` que represente un *tweet*.

3.1. Atributos de clase

Crea, *al menos*, los siguientes **atributos de clase**:

`con`

- Conexión a la base de datos `DB_PATH`.
- *Resultados de consulta* en **modo fila**.

`cur`

- Cursor creado desde la conexión a la base de datos.

3.2. Métodos

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, content: str = '', retweet_from: int = 0, tweet_id: int = 0):
```

- Crea los atributos `retweet_from` e `id` desde los argumentos.
- Crea un atributo `_content` que será la *cadena vacía* si se trata de un *retweet* o tendrá el contenido `content` en otro caso.
- El argumento `retweet_from` indica el *identificador* del *tweet* que se retuitea. Un valor 0 indica que **no** se retuitea.

```
def is_retweet(self) -> bool:
```

- Es una **propiedad**.
- Indica si el *tweet* `self` es un *retweet*.

`def content(self) -> str:`

- Es una **propiedad**.
- Devuelve el contenido del *tweet* `self`.
- Si se trata de un *retweet* el contenido habrá que buscarlo en el *tweet* retuiteado (de la base de datos).

`def save(self, user: User) -> None:`

- Almacena el *tweet* `self` en la base de datos.
- El parámetro `user` indica el usuario que escribió el *tweet*.
- Actualiza –además– el atributo `id` del *tweet* `self` con lo obtenido desde la base de datos después de la inserción.

`def __repr__(self):`

- Devuelve la representación del *tweet* `self` con el siguiente formato:

$$\text{formato} = \begin{cases} \texttt{'<content> (id=<id>)'} & \text{Si es un } \textit{tweet} \\ \texttt{'[RT] <content> (id=<id>)'} & \text{Si es un } \textit{retweet} \end{cases}$$

`def from_db_row(cls, row: sqlite3.Row) -> Tweet:`

- Es un **método de clase**.
- Construye (y devuelve) un objeto de tipo `Tweet` a partir del argumento `row` que es una fila de consulta desde la base de datos.

4. Clase `Twitter`

Escribe una clase `Twitter` que represente el "controlador" de la red social.

4.1. Atributos de clase

Crea, *al menos*, los siguientes **atributos de clase**:

`con`

- Conexión a la base de datos `DB_PATH`.
- *Resultados de consulta* en **modo fila**.

`cur`

- Cursor creado desde la conexión a la base de datos.

4.2. Métodos

Implementa, *al menos*, los siguientes **métodos**:

```
def add_user(self, username: str, password: str, bio: str = '') -> User:
```

- Construye (y devuelve) un objeto de tipo `User` a partir de los argumentos.
- Almacena el objeto creado en la base de datos. Haz uso de la clase `User` y *sus métodos*.
- La contraseña `password` debe cumplir las siguientes reglas:
  1. Empezar con una *arroba* o un *signo igual*.
  2. Continuar con 2, 3 ó 4 *dígitos*.
  3. Continuar con 2, 3 ó 4 *letras* de la `A-Z` (incluyendo minúsculas).
  4. Terminar con una *exclamación* ó un *asterisco*.
- Si la contraseña no sigue este formato debes elevar una excepción de tipo `TwitterError` con el mensaje: `Password does not follow security rules!`

```
def get_user(self, user_id: int) -> User:
```

- Construye (y devuelve) un objeto de tipo `User` a partir de su `user_id`.
- Utiliza la clase `User` y *sus métodos*.
- Si no existe ningún usuario con identificador `user_id` debes elevar una excepción de tipo `TwitterError` con el mensaje: `User with id <user_id> does not exist!`

5. Clase `TwitterError`

Escribe una clase `TwitterError` que represente una excepción y que se pueda usar en el resto del programa para emitir mensajes de error.

