PROGRAMACIÓN III

Clase 9

Clase 8

Manejo de excepciones y Módulos y paquetes

- Manejo de excepciones.
- Errores vs. excepciones.
- Múltiples excepciones, invocación y creación de excepciones propias.
- Módulos y packages.
- Librerías.
- Collections, datetime, math y módulo random.

Clase 9

Base de datos

- Introducción a bases de datos con Python (SQLite,MySQL y SQL).
- Sintaxis básicas.
- Consultas avanzadas. Funciones.
- Conexión a la base de datos.
- Desarrollo y ejecución de operaciones (CRUD).
- Manejo de Archivos.

Clase 10

Django

- ✓ Introducción. **¿Qué es Django?**
- ✓ Instalación
- ✓ **Componentes principales**
- ✓ **Entorno de desarrollo**: Configuración del entorno

MANEJO DE DATOS CON PYTHON

Introducción a bases de datos con Python

En Python, el acceso a bases de datos se encuentra definido a modo de estándar en las especificaciones de DB-API, que puedes leer en la PEP 249. Esto, significa que independientemente de la base de datos que utilicemos, los métodos y procesos de conexión, lectura y escritura de datos, desde Python, siempre serán los mismos, más allá del conector.

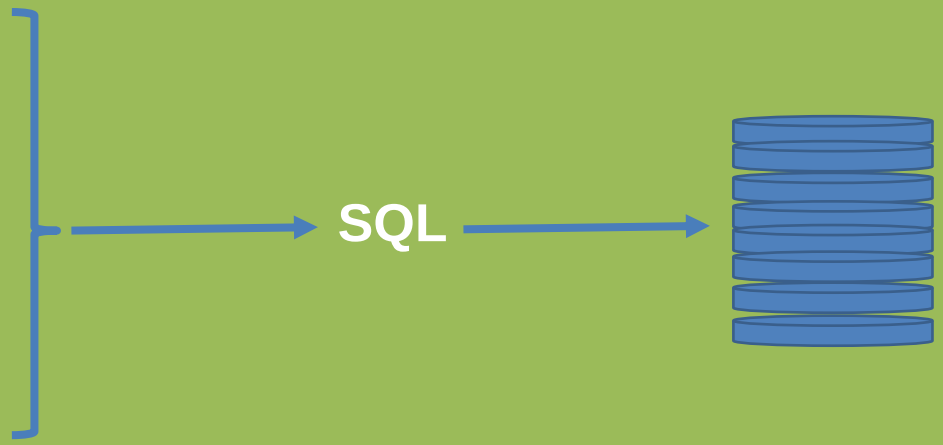
Introducción a bases de datos con Python

La Python **DB API**, actualmente en su versión 2.0, es un conjunto de clases y funciones comunes, estandarizadas, similares para los distintos motores de bases de datos relacionales, escritos en Python. Se desarrolla con la finalidad de lograr la consistencia entre todos estos módulos, y ampliar las posibilidades de crear código portable entre las distintas bases de datos.

SGBD Y PYTHON

Sistemas gestores de bases de datos relacionales o basadas en objetos que manipula python:

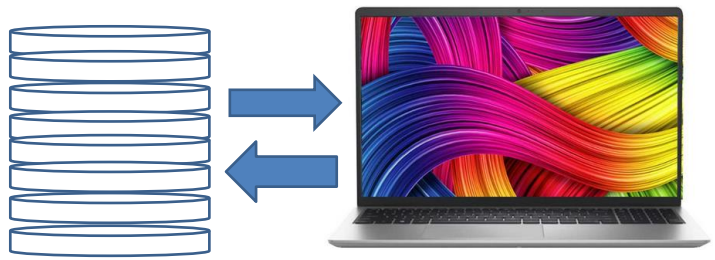
- ✓ SQL SERVER
- ✓ ORACLE
- ✓ MYSQL
- ✓ SQLite
- ✓ PostgreSQL
- ✓ Etc.



SQLite

✓ Que es SQLite

- SQLite es una herramienta de software libre, que permite almacenar información en dispositivos empotrados de una forma sencilla, eficaz, potente, rápida y en equipos con pocas capacidades de hardware, como puede ser una PDA o un teléfono celular.



SQLite

Características

•***Serverless***. **No necesita** una arquitectura cliente/servidor para funcionar. Tampoco necesita de un proceso específico para ejecutarse como un servicio.
•***Single file database***. Cada base de datos se almacena en un único archivo.
•***Zero Configuration***. Al no ser necesario un servidor, no es necesario realizar ninguna configuración adicional. Crear una instancia de una base de datos SQLite es tan sencillo como crear un archivo.
•***Cross-Platform***. El archivo que contiene la base de datos puede ser utilizado en cualquier plataforma (Linux, Windows, macOS).
•***Self-Contained***. La biblioteca SQLite contiene todo el sistema gestor de bases de datos, de modo que se puede integrar fácilmente con la aplicación que haga uso de este sistema.
•***Small Runtime Footprint***. El ejecutable de SQLite ocupa menos de 1 MByte y necesita pocos Megabytes de memoria para ejecutarse.
•***Transactional***. Permite transacciones ACID y permite el acceso seguro a la base de datos desde múltiples procesos y hilos.
•***Full-Featured***. Tiene soporte para la mayoría de características del estándar SQL92 (SQL2)

SQLite Ventajas

1. Es fácil de usar
Ya que no utiliza una comunicación cliente-servidor para las consultas, ya que se comunica con un archivo que es la base de datos y que puede ser autogenerado por la propia aplicación.
2. Ideal para el desarrollo de apps móviles
Se puede utilizar fácilmente para gestionar bases de datos en app que usen motores como Java o Motril, o en proyectos desarrollados con Flutter.
Como la base es un archivo, si se apaga el celular o no hay conexión a internet, elalmacenamiento de datos no se ve afectado.
3. Utiliza SQL
SQLite es una versión reducida de SQL que sigue utilizando este estándar, aunque con pequeñas modificaciones, a la hora de realizar consultas a las bases de datos.
4. Ocupa poco espacio
El almacenamiento de una base de datos SQLite se realiza en un solo archivo y tiene una huella de código pequeña (ocupa poco espacio). En comparación con MySQL, SQLite es una alternativa mucho más ligera, por lo que puede ser utilizada como *software* integrado en dispositivos como celulares, Smart TV, cámaras…

SQLite - Desventajas

•**No es fácilmente escalable**. No se adapta bien a grandes bases de datos, por lo que si una app comienza a crecer se complica su gestión utilizando SQLite.
•**Problemas de seguridad**. Al no contar con funciones de seguridad y administración de usuarios puede presentar problemas en cuanto a seguridad.
•**Monousuario**. No permite que un usuario modifique datos, si otro se encuentra conectado y realizando acciones sobre la base de datos.
•**Limitación de almacenamiento**. El tamaño de la base de datos se encuentra restringido a 2 GB (no es ideal para grandes bases de datos).
•**No admite clausulas anidadas**(where)
•**Falta clave foránea** cuando se crea en modo consola

MySQL

Es un sistema open source de administración de bases de datos que es desarrollado y actualmente es soportado por Oracle.
MySQL fue originalmente lanzado en 1995. Desde entonces, ha pasado por varios cambios de propiedad/administración, antes de terminar en la Oracle Corporation en 2010. A pesar de que Oracle está a cargo ahora, MySQL sigue siendo un **software open source.**
El nombre viene al juntar “My” – el nombre de la hija del co-fundador – con SQL – la abreviatura de Structured Query Language, el cual es el lenguaje de programación que le ayuda a acceder y administrar datos en una base de datos relacional.

MySQL - Ventajas

Facilidad de uso y aprendizaje
Su sintaxis simple y su amplia documentación hacen que sea relativamente fácil comenzar a trabajar con MySQl
Compatibilidad multiplataforma
MySQL es compatible con una amplia variedad de plataformas, incluyendo Windows, Linux y macOS.
Escalabilidad y rendimiento
Su arquitectura optimizada y su capacidad para utilizar múltiples hilos de ejecución permiten un procesamiento eficiente de consultas y transacciones, lo que lo hace adecuado para aplicaciones de alto tráfico.
Amplia comunidad y soporte
MySQL cuenta con una gran comunidad de usuarios y desarrolladores en todo el mundo.
Costo y licencia
MySQL es una base de datos de código abierto, lo que significa que es de uso gratuito y está disponible bajo la Licencia Pública General de GNU (GPL

MySQL - Desventajas

1. Limitaciones de almacenamiento y tamaño de base de datos
Las empresas que manejan grandes volúmenes de datos pueden encontrar restricciones en cuanto a escalabilidad y capacidad de almacenamiento.
2. Funcionalidades avanzadas limitadas
Algunas características más sofisticadas y complejas presentes en otros sistemas pueden no estar disponibles o tener un soporte limitado en MySQL.
3. Replicación y alta disponibilidad
Aunque MySQL ofrece capacidades de replicación y alta disponibilidad, configurar y administrar estos sistemas puede ser complejo y requerir un conocimiento profundo de la tecnología.
4. Optimización de consultas
Aunque MySQL ofrece herramientas y técnicas para mejorar el rendimiento de las consultas, es necesario comprender bien la estructura de la base de datos y tener conocimientos sólidos sobre el motor de almacenamiento utilizado.
5. Transacciones y bloqueo de tablas
MySQL utiliza bloqueo de tablas para gestionar las transacciones, lo que puede resultar en cuellos de botella y tiempos de respuesta más lentos en aplicaciones con una gran cantidad de transacciones concurrentes. Esto puede afectar el rendimiento y la escalabilidad en entornos de alta demanda.

MariaDB

MariaDB es un sistema de gestión de bases de datos relacionales (RDBMS) gratuito y de código abierto. Fue creado por los desarrolladores originales de MySQL por la preocupación de que MySQL pasara a ser comercializado después de que Oracle lo adquiriera en 2009.
MariaDB está escrito en C y C++ y es compatible con varios lenguajes de programación, incluidos C, C#, Java, Python, PHP y Perl.
MariaDB también es compatible con todos los principales sistemas operativos, incluidos Windows, Linux y macOS.
Aunque es una base de datos relacional, MariaDB ofrece funciones similares a las de NoSQL en la versión 10.

MariaDB - Ventajas

1. **Compatibilidad con MySQL**:Esto significa que se pueden realizar fácilmente migraciones desde MySQL a MariaDB sin problemas lo que facilita la transición para aquellos que ya utilizan MySQL como base de datos.
2. **Mejor rendimiento**: MariaDB ofrece mejoras significativas en términos de rendimiento en comparación con MySQL. Esto se debe a la incorporación de nuevas características como la optimización de consultas y el uso de almacenamiento en caché mejorado. Además, MariaDB ha implementado un hilo de ejecución por conexión que reduce la sobrecarga del sistema y mejora el rendimiento general.
3. **Mayor estabilidad**: MariaDB ofrece una mayor estabilidad en comparación con MySQL. Esto se debe en parte a su capacidad para manejar grandes cargas de trabajo y a sus características adicionales de Recuperación ante fallas.
4. **Mayor seguridad:** MariaDB ha corregido y mejorado muchas de las deficiencias de seguridad presentes en MySQL, lo que ayuda a garantizar la integridad de los datos almacenados.
5. **Comunidad activa**: MariaDB cuenta con una comunidad de usuarios muy activa que contribuye al desarrollo y mejora continua del sistema.

MariaDB - Desventajas

1. **Curva de aprendizaje**: Aunque la compatibilidad con MySQL facilita la migración, aún puede haber una curva de aprendizaje para aquellos que no están familiarizados con MariaDB. Esto implica que los administradores de bases de datos y los desarrolladores pueden necesitar tiempo adicional para adaptarse a la nueva plataforma.
2. **Menor soporte de herramientas y plugins**: Aunque MariaDB es compatible con la mayoría de las herramientas y plugins de MySQL, es posible que no todos funcionen perfectamente o estén disponibles para MariaDB. Esto puede suponer una limitación para aquellos que dependen de herramientas específicas o de ciertos plugins para su trabajo diario.
3. **Falta de funciones avanzadas**: Aunque MariaDB ha introducido mejoras significativas en términos de rendimiento y estabilidad, aún puede carecer de algunas funciones avanzadas presentes en otros sistemas de gestión de bases de datos. Esto puede ser un inconveniente para aquellos que necesitan características específicas y sofisticadas en sus proyectos.

¿Cómo podemos trabajar con MySQL/MariaDB desde python3?

Tenemos varios paquetes que nos permiten gestionar una base de datos mysql/mariadb desde python3:

- **mysqlclient**: Es un fork del proyecto MySQLdb1, basado en una librería escrita en C, que implementa la API de acceso a las base de datos MySQL. Tiene soporte para python3 y arregla algunos errores del proyecto original. Podemos usar el módulo MySQLdb escrito en python que nos facilita su uso.
- **PyMySQL**: Librería escrita totalmente en Python que implementa la API de acceso a la base de datos.
- **mysql-connector-python**: El driver (conector) ofrecido oficialmente por Oracle.

Conectarse a la base de datos y ejecutar consultas

Para conectarnos a la base de datos y ejecutar cualquier consulta, el procedimiento consiste en:

1. Abrir la conexión.
2. Crear un cursor
    1. Ejecutar la consulta (SQL)
    2. Manejar los resultados de la query (consulta):
        1. Insertar, leer, actualizar y eliminar (Create, Read, Update y Delete)
3. Cerrar el cursor
4. Cerrar la conexión

PASOS PARA CONECTARSE A UNA BASE DE DATOS MySQL

1. Abrir la conexión usando la función connect()
2. Crear el puntero usando la función cursor()
3. Ejecutar la consulta
4. Obtener los resultados de la consulta o confirmar la operación, dependiendo del tipo de consulta lanzada.
5. Cerrar la conexión

OBJETOS DE LA CONEXION

✓ **CURSOR(): retorna un nuevo objeto cursor usando la conexion**.

Los cursores son un índice virtual que declaramos, generalmente, dentro de un proceso para usarlo en un momento preciso y que nos vuelva un determinado valor. Así pues, el cursor se encarga de recorrer toda una lista o conjunto de recursos donde hemos declarado la herramienta.

Atributos:

- **Descripcion**:
- **Rowcount**: números de filas obtenidas después del método **execute**

Metodos:

- **Close()**: cierra el cursor
- **CallProc()**: para ejecutar procedimientos almacenados
- **Execute()**: prepara y ejecuta operaciones o consultas (query sobre la base de datos)
- **Executemany()**: prepara y ejecuta operaciones o consultas sobre la base de datos con una secuencia de parámetros y como resultado o como datos manipula una tupla con los datos.

MÉTODOS DEL OBJETO CURSOR

Métodos:

- **Fetchone**(): Obtiene la siguiente fila de un conjunto de resultados de consulta, devolviendo una sola secuencia o Ninguno cuando no hay más datos disponibles.
- **Fetchmany()**:Obtiene el siguiente conjunto de filas del resultado de una consulta, devolviendo una secuencia de secuencias (por ejemplo, una lista de tuplas). Se devuelve una secuencia vacía cuando no hay más filas disponibles.
- **Fetchall()**: Obtiene todas las filas (restantes) de un resultado de consulta, devolviéndolas como una secuencia de secuencias (por ejemplo, una lista de tuplas).

EL OBJETO CURSOR


```
miConexion =pymysql.connect(
host="localhost",
user="root",
passwd="",
db="BASE_DE_DATOS")
```

El método connect(), es el constructor que crea una conexión a la base de datos, que retorna un objeto miConexion, con los parámetros indicados (host, user, password, database).

METODO EXECUTE()

El método execute prepara una consulta a la base de datos o bien crea una consulta para la creación de una base de datos

```
cursor.execute("CREATE DATABASE productos")
```

METODO CONMIT()

Cuando una consulta altera o modifica (cláusulas como CREATE, INSERT, UPDATE, DELETE, etc.) información o estructura de alguna tabla de la base de datos, es necesario confirmar esos cambios a través de una operación llamada **commit** y que en Python se realiza a través de la función homónima.

```
Cursor.conmit()
```

METODO CLOSE()

Cuando se termina de usar la conexión, ésta debe cerrarse.
Una vez convocado el método close(), la conexión ya no puede ser utilizada, por lo que se lanzará una excepción de error si se intenta realizar alguna operación con la conexión. Lo mismo se aplica a todos los objetos del cursor que intentan utilizar la conexión. Si cierra una conexión sin haber realizado antes los cambios, se producirá un retroceso implícito

```
miConexion.close()
```

Cómo manejar archivos en Python

En cualquier lenguaje de programación, el manejo de archivos es un aspecto importante. Y Python también admite trabajar con archivos en diferentes modos, como lectura y escritura en archivos, y más.
Podemos:

- ✓ Abrir y leer archivos en python
- ✓ Leer líneas de un archivo de texto
- ✓ Escribir y agregar líneas en archivos
- ✓ Utilizar administradores de contexto para trabajar con archivos.

Leer archivos en Python

Para abrir un archivo en Python, puede usar la sintaxis general:

open('file_name','mode')

Donde:

- ✓ **file_name** es el nombre del archivo.
- ✓ **mode** especifica el modo en el que le gustaría abrir el archivo (El modo predeterminado para abrir un archivo es **leer**—indicado por la letra)

Nota: Si el archivo que desea abrir está en el directorio de trabajo actual, puede mencionar solo el nombre del archivo. Si está en otra carpeta de su entorno de trabajo, debe incluir la ruta del archivo.

Usando administradores de contexto

```
with open('lib.txt','r') as f:
        contents = f.read()
        print(contents)
```

Al usar administradores de contexto para trabajar con archivos, no tiene que usar el close() método. Los archivos se cierran automáticamente una vez finalizada la operación de E/S.
Sin embargo, cuando necesite leer archivos de gran tamaño, utilice el **read**. El método, anterior, puede no ser muy eficiente. De hecho, si el archivo de texto es de un tamaño muy grande, es posible que pronto se quede sin memoria. Por eso es posible que desee leer en líneas de solo lectura de un archivo de texto-

MÉTODO READLINE() DE PYTHON

El método readline lee una línea a la vez del archivo.
Después de la primera readline() llamada al método, se imprime
la primera línea del archivo. Y la segunda llamada al readline() El método devuelve la segunda línea del archivo. Esto se debe a que, después de la primera llamada al método, el puntero del archivo se encuentra al comienzo de la segunda línea.

Programa

```
with open('lib.txt','r') as f:
        line = f.readline()
        print(line)
        line = f.readline()
        print(line)
```

Terminal

```
# Output Hello, there!



Here are a few helpful Python libraries:
```

TELL AND SEEK

En Python, puede usar el **tell**() método para obtener la ubicación actual del puntero del archivo.
Y para mover el puntero del archivo a una ubicación específica, puede usar el **seek**() método

Programa

```
with open('lib.txt','r') as f:
  line = f.readline()
  print(line)
  f.seek(0)
  line = f.readline()
  print(line)
```

Terminal

```
# Output
Hello, there!



Hello, there!
```

USO DEL MÉTODO READLINES()

El método devuelve una lista de todas las líneas del archivo.

Programa

```
with open('lib.txt','r') as f:
  lines = f.readlines()
  print(lines)
```

Terminal

```
# Output
['Hello, there!\n', 'Here are a few
helpful Python libraries:\n',
'1) NumPy\n', '2) pandas\n', '3)
matplotlib\n',
'4) seaborn\n', '5) scikit-learn\n', '6)
BeautifulSoup\n',
'7) Scrapy\n', '8) nltk\n', '9) Bokeh\n',
'10) statsmodels\n', '\n']
```

Usando Loop para leer líneas del archivo

Una vez que tenga un objeto de archivo, podemos usar for para iterar a través del contenido del archivo, una línea a la vez e imprimirlos, como se muestra a continuación.

Programa

```
with open('lib.txt','r') as f:
  for line in f:
    print(line, end='')
```

Terminal

```
# Output
Hello, there!
Here are a few helpful Python libraries:
1) NumPy
2) pandas
3) matplotlib
4) seaborn
5) scikit-learn
6) BeautifulSoup
7) Scrapy
8) nltk
9) Bokeh
10) statsmodels
```

CHUKS

En Python, también puede optar por leer el contenido del archivo en términos de pequeños fragmentos.
Si establecemos el **chunk_size** a 50. Esto significa que se leerán los primeros 50 **caracteres** del archivo y también los imprimiremos.
Ahora, llame al **tell()** método en el objeto de archivo f. Puede ver que el puntero del archivo ahora está en la posición 51, que es como se esperaba.

Programa

```
chunk_size = 50
with open('lib.txt','r') as f:
  chunk = f.read(chunk_size)
  print(chunk)
  current = f.tell()
  print(f"Current position of file pointer: {current}")
```

Terminal

```
# Output
Hello, there!
Here are a few helpful Python libraries:
1) NumPy
2) pandas
3) matplotlib
4) seaborn
5) scikit-learn
6) BeautifulSoup
7) Scrapy
8) nltk
9) Bokeh
10) statsmodels
```

AGREGAR CONTENIDO A UN ARCHIVO

Si desea agregar contenido a un archivo, sin sobrescribirlo, se usa el modo **anexar**.

Programa

```
with open('new_file.txt','a') as f:
  f.write('Hello, Python!')
```

ESCRITURA DE UN ARCHIVO EN Python

Para escribir en un archivo de texto en Python, debe abrirlo en el modo de escritura, especificando 'w'.

Programa

```
with open('new_file.txt','w') as f:
  f.write('Hello, Python!')
```

MATERIAL EXTRA

Artículos de interés

Material extra:

- ✓ Gestión de base de datos mysql/mariadb con python3
- ✓ Conectarse a la base de datos y ejecutar consultas
- ✓ Curso de base de datos con Python
- ✓ Manejo de archivos

Videos:

- ✓ BBDD 1
- ✓ BBDD 2
- ✓ BBDD 3
- ✓ BBDD 4