```
import pyodbc
import logging
import os

# Configuración del logger para DEBUG
```

Este fragmento de código en Python está configurado para inicializar el registro (logging) de eventos para una aplicación. Veamos qué hace cada parte:

> *logging.basicConfig(...): Esta función configura el logging con una configuración básica. La configuración básica es útil para configurar rápidamente el logging sin necesidad de crear instancias de loggers, formateadores y manejadores manualmente.
> *filename="funcionamiento.log": Este argumento especifica que los registros (logs) se escribirán en un archivo llamado funcionamiento.log.
> *level=logging.DEBUG: Este argumento establece el nivel de severidad del logging en DEBUG. El nivel DEBUG proporciona la mayor cantidad de información detallada, capturando todos los eventos que ocurren durante la ejecución del programa. Es útil para diagnósticos detallados.
> *logger = logging.getLogger(__name__): Esta línea crea un logger con el nombre del módulo actual. __name__ es una variable especial en Python que representa el nombre del módulo actual en el que se está ejecutando el código. Si este código se encuentra en el módulo principal, __name__ será igual a '__main__'.

En resumen, este código inicializa un sistema de logging que escribe todos los mensajes de nivel DEBUG (y niveles superiores como INFO, WARNING, ERROR y CRITICAL) al archivo funcionamiento.log. Esto es útil para el seguimiento del funcionamiento de un programa y para la depuración de errores.

```
logging.basicConfig(filename="funcionamiento.log",level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

"""
Muestra el contenido de una tabla específica de la base de datos.

Esta función ejecuta una consulta SQL para obtener todos los registros
de la tabla especificada en 'nombre_tabla'. Luego imprime cada fila del resultado.
Si se produce un error durante la ejecución de la consulta, se captura y se registra.

@param handler El cursor de la base de datos que se utilizará para ejecutar la consulta.
@param nombre_tabla El nombre de la tabla cuyo contenido queremos mostrar.

@exception pyodbc.Error Captura cualquier error de la base de datos que pueda surgir durante
la ejecución de la consulta y lo registra.
"""

```
# Función para mostrar el contenido de una tabla de la BD
def mostrarTablaBD(handler, nombre_tabla):
    try:
        handler.execute("SELECT * FROM " + nombre_tabla)
        logger.debug("Se ejecuta la consulta " + nombre_tabla)
        for row in handler.fetchall():
            print(row)
    except pyodbc.Error as e:
        logger.debug("Error en la consulta ", nombre_tabla, ":", e)
        print("Error en la consulta ", nombre_tabla, ":", e)
```

"""
Reinicia la base de datos eliminando y recreando las tablas DetallePedido, Stock y Pedido.

La función comienza eliminando las tablas si existen, comenzando con DetallePedido debido a
sus llaves foráneas, seguido de Stock y luego Pedido. Después de eliminar las tablas, la función
procede a recrear las tablas Stock y Pedido, y finalmente DetallePedido. También inserta diez
elementos iniciales en la tabla Stock. Se registra cada operación utilizando el logger.

@param handler El cursor de la base de datos para ejecutar los comandos SQL.

Notas:
- Se asume que 'handler' es un cursor válido para una conexión de base de datos de pyodbc.
- Esta función puede generar excepciones de pyodbc si hay problemas con las operaciones de la base de datos.
  Estas excepciones deben ser manejadas por la función llamadora.
"""

```
# Función para borrar las tablas, crearlas de nuevo e insertar 10
tuplas predefinidas a la tabla Stock
def resetearBD(handler):
    # Eliminar tabla DetallePedido
    # Hay que eliminar esta primera poque cuneta con foerign keys
    handler.execute("SELECT TABLE_NAME FROM USER_TABLES WHERE
TABLE_NAME='DETALLEPEDIDO'")
    if len(handler.fetchall()) == 1:
        handler.execute("DROP TABLE DETALLEPEDIDO")
        logger.debug("Eliminada tabla detallepedido")
    else:
        logger.debug("No existe la tabla detallepedido")

    # Eliminar tabla Stock
    handler.execute("SELECT TABLE_NAME FROM USER_TABLES WHERE
TABLE_NAME='STOCK'")
    if len(handler.fetchall()) == 1:
        handler.execute("DROP TABLE STOCK")
        logger.debug("Eliminada tabla stock")
    else:
        logger.debug("No existe la tabla stock")

    # Eliminar tabla Pedido
    handler.execute("SELECT TABLE_NAME FROM USER_TABLES WHERE
TABLE_NAME='PEDIDO'")
    if len(handler.fetchall()) == 1:
        handler.execute("DROP TABLE PEDIDO")
        logger.debug("Eliminada tabla pedido")
```

```
    else:
        logger.debug("No existe la tabla pedido")

    # Crear tabla Stock e insertar 10 elementos
    handler.execute("CREATE TABLE STOCK (Cproducto int constraint
pk_stock primary key, Cantidad int)")
    for i in range(1, 11):
        handler.execute("INSERT INTO STOCK VALUES (" + str(i) + "," +
str(i+5) + ")")
    logger.debug("Creada tabla stock e insertados 10 elementos")

    # Crear tabla Pedido
    handler.execute("CREATE TABLE PEDIDO (Cpedido int constraint
pk_pedido primary key, Ccliente int, FechaPedido date)")
    logger.debug("Creada tabla pedido")

    # Crear tabla DetallePedido
    handler.execute("CREATE TABLE DETALLEPEDIDO (Cpedido int, Cproducto
int, Cantidad int, constraint pk_detallepedido primary key (Cpedido,
Cproducto), constraint fk_detallepedido_pedido foreign key (Cpedido)
references PEDIDO(Cpedido), constraint fk_detallepedido_stock foreign
key (Cproducto) references STOCK(Cproducto))")
    logger.debug("Creada tabla detallepedido")
```

"""
Registra un nuevo pedido en la base de datos.

Esta función inserta un nuevo registro en la tabla PEDIDO utilizando los datos proporcionados.
Si la inserción es exitosa, se registra la acción; si falla, se captura la excepción pyodbc.Error,
se registra el error y se muestra un mensaje al usuario.

@param handler El cursor de la base de datos para ejecutar los comandos SQL.
@param cpedido El código del pedido a insertar.
@param ccliente El código del cliente asociado con el pedido.
@param fecha_pedido La fecha del pedido a insertar.

@exception pyodbc.Error Captura cualquier error de la base de datos que pueda surgir durante
la inserción del pedido y lo registra.
"""

```
# Función para dar de alta un pedido en la BD
def darAltaPedido(handler, cpedido, ccliente, fecha_pedido):
    try:
        handler.execute("INSERT INTO PEDIDO VALUES (" + str(cpedido) +
"," + str(ccliente) + "," + fecha_pedido + ")")
        logger.debug("Insertado pedido " + str(cpedido))
    except pyodbc.Error as e:
        logger.debug("Error al insertar pedido " + str(cpedido) + ":",
e)
        print("Error al insertar pedido " + str(cpedido) + ":", e)
```

```
########################### App.py ############################

connection = None

try:
    cnxn = pyodbc.connect('DRIVER={Devart ODBC Driver for
Oracle};Direct=True;Host=oracle0.ugr.es;Service
Name=practbd.oracle0.ugr.es;User ID=x9617109;Password=x9617109')
    handler = cnxn.cursor()
    logger.debug("Se establece el cursor de conexión")
except pyodbc.Error as e:
    logger.debug("Error connecting to Oracle:", e)
    print("Error connecting to Oracle:", e)
    exit()


while True:
    print("\nSelecciona una opción:\n")
    print("1. Borrar y crear tablas de la BD")
    print("2. Dar de alta un pedido")
    print("3. Mostrar contenido de las tablas de la BD")
    print("4. Limpiar Terminal")
    print("5. Salir\n")

    opcion = int(input("Opción: "))
    # os.system('clear')

    if opcion == 1:
        resetearBD(handler)
    elif opcion == 2:
        # Introducir datos del pedido
        cpedido = input("Introduce el código del pedido: ")
        ccliente = input("Introduce el código del cliente: ")
        fecha_pedido = input("Introduce la fecha del pedido
(DD/MM/YYYY): ")

        # Desactivar autocommit para poder hacer rollback
        cnxn.autocommit = False
        handler.execute("SAVEPOINT inicio")
        logger.debug("Se crea el punto de guardado inicio")
        handler.execute("INSERT INTO PEDIDO VALUES (" + str(cpedido) +
"," + str(ccliente) + ",TO_DATE('" + str(fecha_pedido) +
"','DD/MM/YYYY'))")
        logger.debug("Se inserta el pedido " + str(cpedido))
        handler.execute("SAVEPOINT pedido")
        logger.debug("Se crea el punto de guardado pedido")

        opcion_aux = 0

        while opcion_aux != 3 and opcion_aux != 4:
            print("\nSelecciona una opción:\n")
```

```
            print("1. Añadir detalle de producto")
            print("2. Eliminar todos los detalles del producto")
            print("3. Cancelar pedido")
            print("4. Confirmar pedido\n")

            opcion_aux = int(input("Opción: "))

            if opcion_aux == 1:
                print("\n Productos: \n")
                mostrarTablaBD(handler,"STOCK")
                cproducto = input("Introduce el código del producto: ")
                cantidad = input("Introduce la cantidad: ")
                handler.execute("SELECT CANTIDAD FROM STOCK WHERE
CPRODUCTO=" + str(cproducto))

                if handler.fetchall()[0][0] >= int(cantidad):
                    logger.debug("Coge bien la cantidad " +
str(cantidad))
                    handler.execute("INSERT INTO DETALLEPEDIDO VALUES
(" + str(cpedido) + "," + str(cproducto) + "," + str(cantidad) + ")")
                    logger.debug("Se inserta el detalle de pedido " +
str(cpedido) + " " + str(cproducto) + " " + str(cantidad))
                    handler.execute("UPDATE STOCK SET
CANTIDAD=CANTIDAD-" + str(cantidad) + " WHERE CPRODUCTO=" +
str(cproducto))
                    logger.debug("Se actualiza el stock del producto "
+ str(cproducto))
                else:
                    print("No queda esa cantidad en Stock")
                mostrarTablaBD(handler, "DETALLEPEDIDO")
            elif opcion_aux == 2:
                handler.execute("ROLLBACK TO pedido")
                logger.debug("Se hace rollback al punto de guardado
pedido")
                print("\nTabla Pedido:\n")
                mostrarTablaBD(handler, "PEDIDO")
                print("\nTabla DetallePedido:\n")
                mostrarTablaBD(handler, "DETALLEPEDIDO")
                print("\nTabla Stock:\n")
                mostrarTablaBD(handler, "STOCK")
            elif opcion_aux == 3:
                handler.execute("ROLLBACK TO inicio")
                logger.debug("Se hace rollback al punto de guardado
inicio")
            elif opcion_aux == 4:
                cnxn.commit()
                logger.debug("Se hace commit")
                cnxn.autocommit = True

    elif opcion == 3:
        print("\nTabla STOCK:\n")
```

```
        mostrarTablaBD(handler, "STOCK")
        print("\nTabla PEDIDO:\n")
        mostrarTablaBD(handler, "PEDIDO")
        print("\nTabla DETALLEPEDIDO:\n")
        mostrarTablaBD(handler, "DETALLEPEDIDO")
    elif opcion == 5:
        handler.close()
        cnxn.close()
        del handler
        del cnxn
        exit()
    elif opcion == 4:
        os.system("clear")
```

El código proporcionado es un fragmento de un programa en Python que utiliza la biblioteca `pyodbc` para interactuar con una base de datos Oracle. Aquí hay una explicación detallada de lo que hace el código:

- `connection = None`: Esto inicializa la variable `connection` a `None`, lo que probablemente sea un placeholder para su uso posterior en el código.
- Bloque `try-except`: Este bloque intenta establecer una conexión con una base de datos Oracle usando un string de conexión con `pyodbc.connect()`. Si la conexión es exitosa, crea un cursor de base de datos (`handler`) que se usa para ejecutar comandos SQL. Si hay un error durante la conexión, se captura la excepción `pyodbc.Error`, se registra el error y se imprime un mensaje. Luego, el programa se termina con `exit()`.
- Bucle `while True`: Este es un bucle infinito que presenta al usuario un menú con varias opciones relacionadas con la base de datos, como borrar y crear tablas, dar de alta un pedido, mostrar el contenido de las tablas, limpiar la terminal o salir del programa.
- Opción 1: Si el usuario elige la opción 1, se llama a la función `resetearBD(handler)`, la cual asumo que reinicia las tablas en la base de datos según la definición previa proporcionada.
- Opción 2: En esta opción, el usuario introduce los datos de un nuevo pedido. Se desactiva el `autocommit` para iniciar una transacción y se establecen puntos de guardado. Luego, el programa ofrece un submenú para gestionar los detalles del pedido, con opciones para añadir detalles del producto, eliminar todos los detalles, cancelar el pedido o confirmarlo.
- Opción 3: Muestra el contenido de las tablas `STOCK`, `PEDIDO` y `DETALLEPEDIDO` llamando a la función `mostrarTablaBD(handler, nombre_tabla)`.
- Opción 4: Limpia la terminal. En sistemas Unix, `os.system('clear')` borra la pantalla de la terminal.

Opción 5: Cierra el cursor y la conexión a la base de datos, elimina las variables `handler` y `cnxn` para liberar recursos y luego sale del programa con `exit()`.

El código implementa una sencilla interfaz de línea de comandos (CLI) para interactuar con la base de datos y realizar operaciones comunes. Maneja cuidadosamente las transacciones utilizando puntos de guardado y `commit/rollback` para asegurar la consistencia de los datos. Además, utiliza un sistema de logging para registrar los eventos importantes que ocurren durante la ejecución del programa.