perplexity

create the programs with name of the program on top mentioned in commented form and create it in python

Okay, I will create the Python programs as described in the image, with the program name in a comment at the top of each file.

```
# SQLITE DATABASE OPERATIONS:
import sqlite3

# Connect to the database (or create it if it doesn't exist)
conn = sqlite3.connect('students.db')

# Create a cursor object to execute SQL commands
cursor = conn.cursor()

# Create the students table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        grade TEXT
    )
''')

# Insert data into the table
students_data = [
    (1, 'Alice', 'A'),
    (2, 'Bob', 'B'),
    (3, 'Charlie', 'C'),
    (4, 'David', 'B'),
    (5, 'Eve', 'A+')
]

cursor.executemany('INSERT INTO students (id, name, grade) VALUES (?, ?, ?)', students_da

# Retrieve all records from the table
cursor.execute('SELECT * FROM students')
rows = cursor.fetchall()

# Display the records
for row in rows:
    print(row)

# Commit the changes and close the connection
```

```
conn.commit()
conn.close()
```

```
# DATABASE AND EXCEPTION HANDLING:
import sqlite3

try:
    # Connect to the database
    conn = sqlite3.connect('students.db')
    cursor = conn.cursor()

    # Attempt to create the table (if it doesn't exist)
    try:
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS students (
                id INTEGER PRIMARY KEY,
                name TEXT NOT NULL,
                grade TEXT
            )
        ''')
        print("Table created successfully (if it didn't exist).")
    except sqlite3.Error as e:
        print(f"Error creating table: {e}")

    # Attempt to insert data
    try:
        students_data = [
            (1, 'Alice', 'A'),
            (2, 'Bob', 'B'),
            (3, 'Charlie', 'C'),
            (4, 'David', 'B'),
            (5, 'Eve', 'A+')
        ]
        cursor.executemany('INSERT INTO students (id, name, grade) VALUES (?, ?, ?)', stu
        print("Data inserted successfully.")
    except sqlite3.Error as e:
        print(f"Error inserting data: {e}")

    # Attempt to retrieve data
    try:
        cursor.execute('SELECT * FROM students')
        rows = cursor.fetchall()

        for row in rows:
            print(row)
    except sqlite3.Error as e:
        print(f"Error retrieving data: {e}")

    # Commit changes
    try:
        conn.commit()
        print("Changes committed successfully.")
    except sqlite3.Error as e:
        print(f"Error committing changes: {e}")
```

```
except sqlite3.Error as e:
    print(f"Connection error: {e}")
finally:
    if conn:
        conn.close()
        print("Connection closed.")
```

```
# data_handler.py
# MODULES FOR DATA MANIPULATION:

def read_data(file_path):
    """Reads a text file and returns its content."""
    try:
        with open(file_path, 'r') as file:
            return file.read()
    except FileNotFoundError:
        return "File not found."
    except Exception as e:
        return f"An error occurred: {e}"

def write_data(data, file_path):
    """Writes data to a text file."""
    try:
        with open(file_path, 'w') as file:
            file.write(data)
        return "Data written successfully."
    except Exception as e:
        return f"An error occurred: {e}"

if __name__ == '__main__':
    # Demonstrate reading and writing
    file_to_read = 'input.txt'
    file_to_write = 'output.txt'

    # Create a sample input file
    with open(file_to_read, 'w') as f:
        f.write("This is a sample text in input.txt")

    # Read data from the input file
    content = read_data(file_to_read)
    print(f"Content read from {file_to_read}:\n{content}")

    # Write the content to another file
    write_result = write_data(content, file_to_write)
    print(write_result)

    # Verify the content of the output file
    read_output = read_data(file_to_write)
    print(f"Content of {file_to_write}:\n{read_output}")
```

Create a directory named `my_package`. Inside that directory, create two python files: `utilities.py` and `data_handler.py` (the one from the previous step). Also, make sure to add an `__init__.py` file to the `my_package` directory (it can be empty).

```
# my_package/utilities.py
def utility_function():
    return "This is a utility function from my_package."
```

```
# my_package/data_handler.py
# MODULES FOR DATA MANIPULATION:

def read_data(file_path):
    """Reads a text file and returns its content."""
    try:
        with open(file_path, 'r') as file:
            return file.read()
    except FileNotFoundError:
        return "File not found."
    except Exception as e:
        return f"An error occurred: {e}"

def write_data(data, file_path):
    """Writes data to a text file."""
    try:
        with open(file_path, 'w') as file:
            file.write(data)
        return "Data written successfully."
    except Exception as e:
        return f"An error occurred: {e}"
```

```
# PACKAGE CREATION:
# main.py
#Demonstrates how to import and use functions from the my_package package

import sys
sys.path.append('./my_package')  # Add the package directory to the Python path

from my_package import data_handler, utilities

# Use the functions from the package
file_content = data_handler.read_data('input.txt')
print(f"Content from file: {file_content}")

utility_message = utilities.utility_function()
print(utility_message)
```

```
# ADVANCED LAMBDA AND EXCEPTION HANDLING:
# Lambda and Exception Handling

def inverse(x):
    """
    Returns the inverse of a number using a lambda function.
    Handles exceptions for zero and non-numeric inputs.
    """
    try:
        # Define a lambda function to calculate the inverse
```

```
        inverse_func = lambda a: 1 / float(a)

        return inverse_func(x)
    except ZeroDivisionError:
        return "Cannot divide by zero!"
    except (TypeError, ValueError):
        return "Invalid input: Must be a number!"

# Test the lambda function with a list of values
test_values = [2, 0, 'hello', 5.5, -10, [1,2]]

for value in test_values:
    result = inverse(value)
    print(f"The inverse of {value} is: {result}")
```

Explanation and How to Run the Code:

1. **SQLITE DATABASE OPERATIONS:**
    - This script connects to an SQLite database named `students.db`. If the database doesn't exist, it will be created.
    - It creates a table named `students` with columns `id`, `name`, and `grade`.
    - It inserts five sample records into the table.
    - It retrieves all records from the table and prints them to the console.
    - To run: Save the code as a `.py` file (e.g., `sqlite_operations.py`) and run it from your terminal: `python sqlite_operations.py`
2. **DATABASE AND EXCEPTION HANDLING:**
    - This script builds upon the previous one by adding exception handling. It wraps the database operations in `try...except` blocks to catch potential errors such as connection errors, table creation errors, insertion errors, and retrieval errors.
    - It provides specific error messages for different types of exceptions.
    - To run: Save the code as a `.py` file (e.g., `sqlite_exception.py`) and run it from your terminal: `python sqlite_exception.py`
3. **MODULES FOR DATA MANIPULATION (`data_handler.py`):**
    - This script creates a module named `data_handler.py` with two functions: `read_data` and `write_data`.
    - `read_data(file_path)` reads the content of a text file and returns it as a string. It includes error handling for `FileNotFoundError` and other potential exceptions.
    - `write_data(data, file_path)` writes the given data to a text file. It also includes error handling.
    - The `if __name__ == '__main__':` block demonstrates how to use the functions to read from one file (`input.txt`) and write to another (`output.txt`). It creates a sample `input.txt` file.
    - To run: Save the code as `data_handler.py` and then run it from your terminal: `python data_handler.py`. Make sure `input.txt` is in the same directory, or update the file paths

accordingly.

4. **PACKAGE CREATION:**
    - This part involves creating a package named `my_package` and organizing the `data_handler.py` file into it.
    - Create a new directory named `my_package`.
    - Move the `data_handler.py` file into the `my_package` directory.
    - Inside the `my_package` directory, create an empty file named `__init__.py`. This file is required to make Python treat the directory as a package.
    - Create `utilities.py` inside the `my_package` directory.
    - The `main.py` script demonstrates how to import and use the functions from the `my_package` package. It adds the package directory to the Python path using `sys.path.append()`. Then, it imports the `data_handler` module and calls the `read_data()` function. It also imports `utilities` and calls the `utility_function()` function.
    - To run: Save the `main.py` file in a directory *outside* the `my_package` directory. Create an `input.txt` file in the same directory as `main.py`. Then, run `main.py` from your terminal: `python main.py`
5. **ADVANCED LAMBDA AND EXCEPTION HANDLING:**
    - This script defines a function `inverse(x)` that uses a lambda function to calculate the inverse of a number (1/x).
    - It includes exception handling to manage cases where the input is zero (which would cause a `ZeroDivisionError`) or not a number (which would cause a `TypeError` or `ValueError`).
    - It tests the function with a list of numeric and non-numeric values and prints the results.
    - To run: Save the code as a `.py` file (e.g., `lambda_exception.py`) and run it from your terminal: `python lambda_exception.py`

These programs should fulfill the requirements outlined in the image. Let me know if you have any other questions.

⁂