**AI ASSISTED CODING**

ASSIGNMENT – 10.4

NAME : N.KEERTHI REDDY

HALL.NO : 2403A52376

BATCH : AI 14

TASK – 1

CODE:

A screenshot of a computer code

AI-generated content may be incorrect.

OUTPUT :



CODE EXPLANATION :

1. **def add\_numbers(a, b):**: This line defines the function named add\_numbers and specifies that it accepts two parameters, a and b.
2. **result = a + b**: Inside the function, this line adds the values of a and b and assigns the sum to a variable named result.
3. **return result**: This line returns the value stored in the result variable, which is the sum of a and b.
4. **print(add\_numbers(10, 20))**: This line calls the add\_numbers function with the arguments 10 and 20. The function returns 30, which is then printed to the console.

**TASK – 2**

CODE :

A screen shot of a computer code

AI-generated content may be incorrect.

OUTPUT :



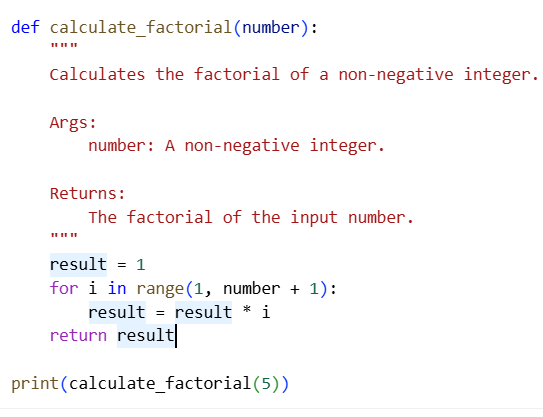
CODE EXPLANATION :

1. **def find\_duplicates\_optimized(nums):**: This line defines the function named find\_duplicates\_optimized and specifies that it accepts one parameter, nums, which is expected to be a list of numbers.
2. **seen = set()**: This line initializes an empty set called seen. This set will be used to keep track of the numbers encountered so far. Using a set for this purpose allows for very fast lookups.
3. **duplicates = set()**: This line initializes an empty set called duplicates. This set will store the numbers that are found to be duplicates.
4. **for num in nums:**: This line starts a loop that iterates through each number in the input list nums.
5. **if num in seen:**: Inside the loop, this line checks if the current number (num) is already present in the seen set. If it is, it means this is a duplicate number.
6. **duplicates.add(num)**: If the number is found in the seen set (i.e., it's a duplicate), this line adds the number to the duplicates set.
7. **seen.add(num)**: After checking for duplicates, this line adds the current number (num) to the seen set. This ensures that the next time this number is encountered, it will be identified as a duplicate.
8. **return list(duplicates)**: After the loop finishes, this line converts the duplicates set into a list and returns it.
9. **numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]**: This line creates a list of numbers to be used as input for the function.

10**.print(find\_duplicates\_optimized(numbers))**: This line calls the find\_duplicates\_optimized function with the numbers list and prints the returned list of duplicates to the console.

**TASK -3**

CODE :



OUTPUT :

A black and white image of a number

AI-generated content may be incorrect.

CODE EXPLANATION :

1. **def calculate\_factorial(number):**: This line defines the function named calculate\_factorial and specifies that it accepts one parameter, number, which is expected to be a non-negative integer.
2. **""" ... """**: This is a docstring, which provides a brief explanation of what the function does, its arguments, and what it returns.
3. **result = 1**: This line initializes a variable called result to 1. This is because the factorial of 0 is 1, and it also serves as the starting point for the multiplication in the loop.
4. **for i in range(1, number + 1):**: This line starts a for loop that iterates from 1 up to and including the value of number.
5. **result = result \* i**: Inside the loop, this line multiplies the current value of result by the current value of i and updates result with the new value. This repeatedly multiplies result by each integer from 1 up to number.
6. **return result**: After the loop finishes, this line returns the final value of result, which is the factorial of the input number.
7. **print( calculate\_factorial(5))**: This line calls the calculate\_factorial function with the argument 5 and prints the returned value (which is 120, the factorial of 5) to the console.

**TASK – 4**

CODE :

# enhanced\_code\_task4.py

import sqlite3

def get\_user\_data\_enhanced(user\_id):

"""

Retrieves user data from the database with enhanced security and error handling.

Args:

user\_id: The ID of the user to retrieve.

Returns:

The user data if found, otherwise None.

"""

conn = None # Initialize connection to None

try:

# Input validation: Check if user\_id is a digit

if not str(user\_id).isdigit():

print("Invalid user ID. Please enter a numeric ID.")

return None

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

# Use parameterized query to prevent SQL injection

query = "SELECT \* FROM users WHERE id = ?;"

cursor.execute(query, (user\_id,))

result = cursor.fetchall()

return result

except sqlite3.Error as e:

print(f"Database error: {e}")

return None

except Exception as e:

print(f"An unexpected error occurred: {e}")

return None

finally:

if conn:

conn.close()

# Example usage:

# Create a dummy database and table for demonstration

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("DROP TABLE IF EXISTS users;")

cursor.execute("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT);")

cursor.execute("INSERT INTO users (name) VALUES ('Alice');")

cursor.execute("INSERT INTO users (name) VALUES ('Bob');")

conn.commit()

conn.close()

user\_input = input("Enter user ID: ")

user\_data = get\_user\_data\_enhanced(user\_input)

if user\_data:

print("User Data:", user\_data)

else:

print("User not found or invalid input.")

OUTPUT :

A close up of a text

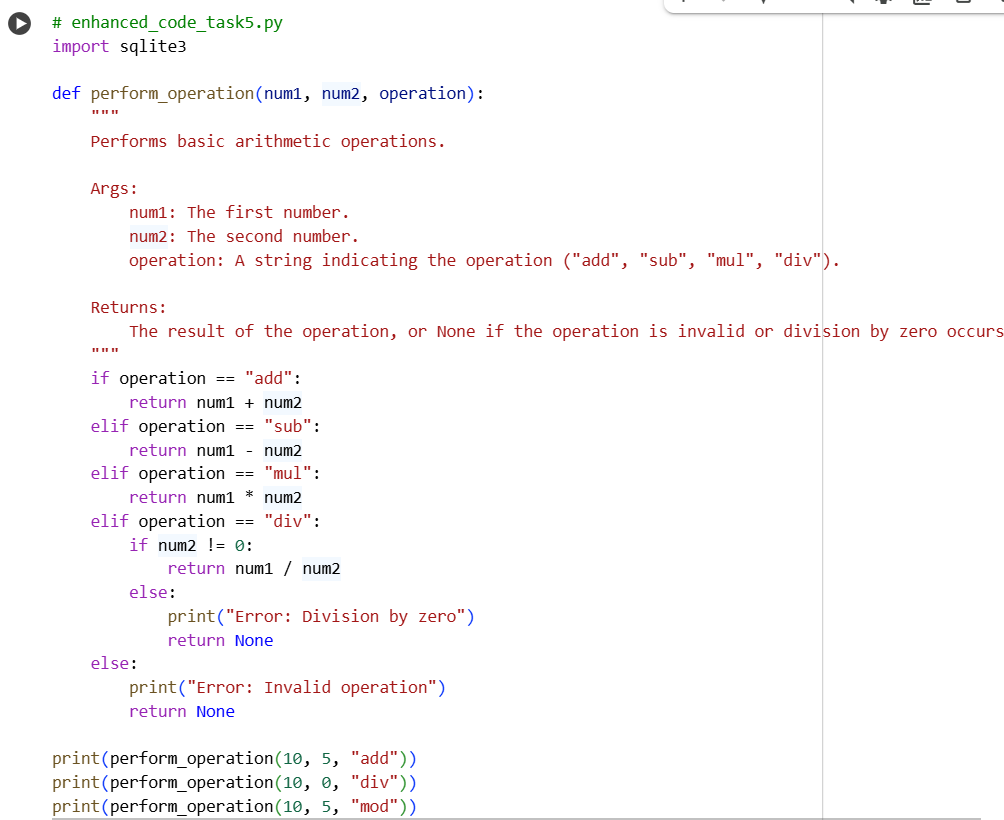
AI-generated content may be incorrect.

CODE EXPLANATION :

1. **import sqlite3**: This line imports the sqlite3 library, which provides an interface for working with SQLite databases.
2. **def get\_user\_data\_enhanced(user\_id):**: This defines the function get\_user\_data\_enhanced that takes one argument, user\_id.
3. **""" ... """**: This is a docstring explaining the function's purpose, arguments, and return value.
4. **conn = None**: Initializes the database connection variable conn to None. This is good practice for error handling in the finally block.
5. **try:**: This block starts a try...except...finally block to handle potential errors during database operations.
6. **if not str(user\_id).isdigit():**: This performs input validation. It checks if the user\_id can be converted to a digit string. If not, it prints an error message and returns None.
7. **conn = sqlite3.connect("users.db")**: This line establishes a connection to the SQLite database file named "users.db". If the file doesn't exist, it will be created.
8. **cursor = conn.cursor()**: This creates a cursor object, which is used to execute SQL commands.
9. **query = "SELECT \* FROM users WHERE id = ?;"**: This defines the SQL query to select all columns from the users table where the id matches the provided user\_id. The ? is a placeholder for a parameterized query, which helps prevent SQL injection.
10. **cursor.execute(query, (user\_id,))**: This executes the SQL query. The user\_id is passed as a tuple (user\_id,) to the execute method, which safely substitutes the placeholder.
11. **result = cursor.fetchall()**: This fetches all rows from the result of the query and stores them in the result variable.
12. **return result**: The function returns the fetched user data (or an empty list if no user is found).
13. **except sqlite3.Error as e:**: This block catches any sqlite3.Error exceptions that occur during the database operations, prints an error message including the specific error, and returns None.
14. **except Exception as e:**: This block catches any other unexpected exceptions, prints a generic error message, and returns None.
15. **finally:**: This block is always executed, regardless of whether an exception occurred or not.
16. **if conn:**: This checks if the conn variable is not None (meaning the connection was successfully established).
17. **conn.close()**: If the connection exists, this line closes the database connection.
18. The code outside the function sets up a dummy database and table named users with two entries ('Alice' and 'Bob') for demonstration purposes.
19. **user\_input = input("Enter user ID: ")**: This line prompts the user to enter a user ID.
20. **user\_data = get\_user\_data\_enhanced(user\_input)**: This calls the get\_user\_data\_enhanced function with the user's input.
21. **if user\_data:**: This checks if the function returned any data.
22. **print("User Data:", user\_data)**: If data is returned, it prints the user data.
23. **else: print("User not found or invalid input.")**: If no data is returned (either because the user was not found or the input was invalid), it prints a corresponding message.

**TASK – 5**

CODE :



OUTPUT :

A screenshot of a computer

AI-generated content may be incorrect.

CODE EXPLANATION :

1. **# enhanced\_code\_task5.py**: This is a comment indicating the filename, though it doesn't affect the code's execution.
2. **import sqlite3**: This line imports the sqlite3 library. While imported, it's not actually used in the perform\_operation function itself. It might be leftover from a previous context or intended for future expansion.
3. **def perform\_operation(num1, num2, operation):**: This defines the function perform\_operation that accepts three arguments: num1 (the first number), num2 (the second number), and operation (a string specifying the operation).
4. **""" ... """**: This is a docstring that explains the function's purpose, arguments, and what it returns.
5. **if operation == "add":**: This is the start of an if-elif-else block that checks the value of the operation string. If it's "add", it proceeds to the next line.
6. **return num1 + num2**: If the operation is "add", this line returns the sum of num1 and num2.
7. **elif operation == "sub":**: If the operation is not "add", this checks if it's "sub".
8. **return num1 - num2**: If the operation is "sub", this line returns the difference between num1 and num2.
9. **elif operation == "mul":**: If the operation is neither "add" nor "sub", this checks if it's "mul".
10. **return num1 \* num2**: If the operation is "mul", this line returns the product of num1 and num2.
11. **elif operation == "div":**: If the operation is none of the above, this checks if it's "div".
12. **if num2 != 0:**: Inside the "div" block, this nested if statement checks if the second number (num2) is not equal to 0. This is crucial for preventing division by zero errors.
13. **return num1 / num2**: If the operation is "div" and num2 is not 0, this line returns the result of dividing num1 by num2.
14. **else:**: This else block is executed if the operation is "div" but num2 is 0.
15. **print("Error: Division by zero")**: This line prints an error message to the console indicating division by zero.
16. **return None**: After printing the error, this line returns None to indicate that the division operation could not be completed successfully.
17. **else:**: This else block is executed if the operation string does not match any of the valid operations ("add", "sub", "mul", "div").
18. **print("Error: Invalid operation")**: This line prints an error message indicating that the provided operation is invalid.
19. **return None**: After printing the error, this line returns None to indicate an invalid operation.
20. **print(perform\_operation(10, 5, "add"))**: These lines demonstrate how to call the perform\_operation function with different arguments and print the returned results. The first call performs addition (10 + 5 = 15).
21. **print(perform\_operation(10, 0, "div"))**: This call attempts division by zero, which triggers the error handling for that case, printing an error message and returning None.
22. **print(perform\_operation(10, 5, "mod"))**: This call uses an invalid operation ("mod"), which triggers the error handling for invalid operations, printing an error message and returning None.