2303A51415

BATCH 03

1. ASSIGNMENT 5.5

```
def is_prime_naive(n):
    """
    Checks if a number is prime using a naive approach.
    Tests divisibility from 2 up to n-1.
    """
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

# Test cases
print(f"Is 7 prime (naive)? {is_prime_naive(7)}")
print(f"Is 10 prime (naive)? {is_prime_naive(10)}")
print(f"Is 2 prime (naive)? {is_prime_naive(2)}")
print(f"Is 1 prime (naive)? {is_prime_naive(1)}")
```

OUTPUT:

```
Is 7 prime (naive)? True
Is 10 prime (naive)? False
Is 2 prime (naive)? True
Is 1 prime (naive)? False
import math

def is_prime_optimized(n):
    """
    Checks if a number is prime using an optimized approach.
    Tests divisibility up to sqrt(n) and uses 6k +/- 1 optimization.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True  # 2 and 3 are prime
    if n % 2 == 0 or n % 3 == 0:
        return False
```

```
    # Check for divisors from 5 up to sqrt(n)
    # We can check numbers of the form 6k +/- 1
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

# Test cases
print(f"Is 7 prime (optimized)? {is_prime_optimized(7)}")
print(f"Is 10 prime (optimized)? {is_prime_optimized(10)}")
print(f"Is 2 prime (optimized)? {is_prime_optimized(2)}")
print(f"Is 1 prime (optimized)? {is_prime_optimized(1)}")
print(f"Is 97 prime (optimized)? {is_prime_optimized(97)}")
OUTPUT:
Is 7 prime (optimized)? True
Is 10 prime (optimized)? False
Is 2 prime (optimized)? True
Is 1 prime (optimized)? False
Is 97 prime (optimized)? True
```

```
import time

def benchmark_prime_check(func, num_to_check):
    start_time = time.time()
    result = func(num_to_check)
    end_time = time.time()
    return result, (end_time - start_time) * 1000 # Convert to
milliseconds

# Choose a relatively large number for benchmarking
large_number = 999999937  # A large prime number

print(f"Checking if {large_number} is prime:")

result_naive, time_naive = benchmark_prime_check(is_prime_naive,
large_number)
print(f"Naive Approach: Prime={result_naive}, Time Taken: {time_naive:.4f}
ms")
```

```
result_optimized, time_optimized =
benchmark_prime_check(is_prime_optimized, large_number)
print(f"Optimized Approach: Prime={result_optimized}, Time Taken:
{time_optimized:.4f} ms")

print(f"\nOptimized approach was approximately {time_naive /
time_optimized:.2f} times faster than the naive approach.")
OUTPUT:
Checking if 999999937 is prime:
Naive Approach: Prime=True, Time Taken: 78326.8461 ms
Optimized Approach: Prime=True, Time Taken: 0.9289 ms

Optimized approach was approximately 84324.08 times faster than the naive
approach.
```

QUESTION 2

```
def fibonacci_recursive(n):
    """
    Calculates the nth Fibonacci number using a recursive approach.

    Args:
        n (int): The index of the Fibonacci number to calculate
(non-negative).

    Returns:
        int: The nth Fibonacci number.
    """
    # Base Cases:
    # These are the conditions where the function returns a value directly
    # without making further recursive calls. They are essential to stop
    # the recursion and prevent infinite loops.
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    # Recursive Calls:
    # This is the step where the function calls itself with smaller
inputs.
    # Each call reduces 'n' by 1 or 2, eventually reaching one of the base
cases.
```

```
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

# Test cases to demonstrate the function
print(f"fibonacci_recursive(0): {fibonacci_recursive(0)}") # Expected: 0
print(f"fibonacci_recursive(1): {fibonacci_recursive(1)}") # Expected: 1
print(f"fibonacci_recursive(2): {fibonacci_recursive(2)}") # Expected: 1
print(f"fibonacci_recursive(3): {fibonacci_recursive(3)}") # Expected: 2
print(f"fibonacci_recursive(6): {fibonacci_recursive(6)}") # Expected: 8
print(f"fibonacci_recursive(10): {fibonacci_recursive(10)}") # Expected:
55
```

OUTPUT:

```
fibonacci_recursive(0): 0
fibonacci_recursive(1): 1
fibonacci_recursive(2): 1
fibonacci_recursive(3): 2
fibonacci_recursive(6): 8
fibonacci_recursive(10): 55
```

Task 03

```
import os

def process_file_with_error_handling(filepath):
    """
    Reads a file and attempts to process its content, with robust error
handling.
    For demonstration, it tries to read lines and sum numbers if possible.

    Args:
        filepath (str): The path to the file to be processed.

    Returns:
        bool: True if the file was processed successfully, False
otherwise.
    """
    print(f"\nAttempting to process file: '{filepath}'")
    try:
        # Attempt to open and read the file
        with open(filepath, 'r', encoding='utf-8') as f:
            content = f.readlines()
```

```
            print(f"Successfully read {len(content)} lines from
'{filepath}'.")

            # Example processing: try to sum numbers if lines contain them
            total_sum = 0
            is_numeric_data = False
            for i, line in enumerate(content):
                try:
                    num = float(line.strip())
                    total_sum += num
                    is_numeric_data = True
                except ValueError:
                    # Ignore lines that are not numbers for this specific
processing
                    pass
            if is_numeric_data:
                print(f"Sum of numeric values in the file:
{total_sum:.2f}")
            else:
                print("No numeric data found or processed in the file.")

        return True

    except FileNotFoundError:
        # This exception is raised if the specified file does not exist.
        print(f"Error: The file '{filepath}' was not found. Please check
the path and filename.")
        return False
    except PermissionError:
        # This exception is raised if the program does not have the
necessary
        # permissions to read the file (e.g., file is locked, or access
denied).
        print(f"Error: Permission denied to access '{filepath}'. Please
check file permissions.")
        return False
    except IsADirectoryError:
        # This exception is raised if the provided path is a directory,
not a file.
```

```
        print(f"Error: '{filepath}' is a directory, not a file. Please
provide a file path.")
        return False
    except UnicodeDecodeError as e:
        # This exception occurs if the file content cannot be decoded
using the
        # specified encoding (e.g., 'utf-8'). This is common with binary
files
        # or text files saved with a different encoding.
        print(f"Error: Could not decode file '{filepath}' with 'utf-8'
encoding. "
              f"It might be a binary file or have a different text
encoding. Details: {e}")
        return False
    except IOError as e:
        # This is a general I/O error, catching other input/output related
problems
        # that are not covered by the more specific exceptions above.
        print(f"Error: An I/O error occurred while processing
'{filepath}'. Details: {e}")
        return False
    except Exception as e:
        # This is a general catch-all for any other unexpected errors that
might occur
        # during file operations or initial processing. It's good practice
to catch
        # specific exceptions first and then a general one as a fallback.
        print(f"Error: An unexpected error occurred while processing
'{filepath}'. Details: {e}")
        return False
# Scenario 1: FileNotFoundError (File does not exist)
non_existent_file = "non_existent.txt"
process_file_with_error_handling(non_existent_file)
```

Output:

```
Attempting to process file: 'non_existent.txt'
Error: The file 'non_existent.txt' was not found. Please check the path
and filename.

False
```

```
# Scenario 2: Successful processing (File exists with valid content)
# Create a dummy file for successful processing
with open('valid_data.txt', 'w', encoding='utf-8') as f:
    f.write('Line 1\n')
    f.write('10.5\n')
    f.write('Line 3 with text\n')
    f.write('20\n')

process_file_with_error_handling('valid_data.txt')

# Clean up the dummy file
os.remove('valid_data.txt')
```

Output:

```
Attempting to process file: 'valid_data.txt'
Successfully read 4 lines from 'valid_data.txt'.
Sum of numeric values in the file: 30.50
# Scenario 3: IsADirectoryError (Path points to a directory)
# Create a dummy directory
if not os.path.exists('dummy_dir'):
    os.makedirs('dummy_dir')

process_file_with_error_handling('dummy_dir')

# Clean up the dummy directory
os.rmdir('dummy_dir')
```

Output:

```
Attempting to process file: 'dummy_dir'
Error: 'dummy_dir' is a directory, not a file. Please provide a file path.
```

Scenario 4: PermissionError (Attempting to open a file without read permissions)

```
# This scenario can be tricky to simulate portably without admin rights.
# On Linux/macOS, you could create a file with no read permissions (chmod 000 file.txt)
# On Windows, it's harder to programmatically deny read access for the current user.
# We will simulate by trying to read a path that might be protected (e.g., root directory without sufficient permissions).
```

```
# This might still raise FileNotFoundError or IsADirectoryError depending
on OS and user.

# For demonstration, we'll try to access a typical protected system
directory.
# Note: This might not always trigger PermissionError if running with
elevated privileges
# or if the OS handles it differently.
protected_path = '/root' # Example for Linux/macOS
if os.name == 'nt': # For Windows
    protected_path = 'C:\\Windows\\System32\\config'

print(f"\nAttempting to trigger PermissionError (or similar) with:
{protected_path}")
process_file_with_error_handling(protected_path)

# A more robust way to test PermissionError often requires setting actual
permissions
# or trying to open a file that is already locked by another process.
# For a simple runnable example, the above might suffice to show a
security-related issue.

# Let's try creating a file and then making it unreadable (if OS allows)
# This part might fail on some systems depending on user permissions.
if os.name != 'nt': # Linux/macOS specific permission change
    permission_denied_file = 'no_read_permission.txt'
    with open(permission_denied_file, 'w') as f:
        f.write('secret data')
    try:
        os.chmod(permission_denied_file, 0o000) # Remove all permissions
        process_file_with_error_handling(permission_denied_file)
    except Exception as e:
        print(f"Could not set permissions or test: {e}")
    finally:
        os.chmod(permission_denied_file, 0o644) # Restore permissions for
cleanup
        os.remove(permission_denied_file)
else:
    print("Skipping explicit PermissionError test for Windows due to OS
specifics.")
```

```
Attempting to trigger PermissionError (or similar) with: /root

Attempting to process file: '/root'
Error: '/root' is a directory, not a file. Please provide a file path.
output:
Attempting to process file: 'no_read_permission.txt'
Successfully read 1 lines from 'no_read_permission.txt'.
No numeric data found or processed in the file.
# Scenario 5: UnicodeDecodeError (Attempting to read a binary file as
text)
# Create a dummy binary file (e.g., a tiny PNG-like header)
with open('binary_file.bin', 'wb') as f:
    f.write(b'\x89PNG\x0d\x0a\x1a\x0a\x00\x00\x00\x0dIHDR')

process_file_with_error_handling('binary_file.bin')

# Clean up the dummy file
os.remove('binary_file.bin')


Task 04
import json
import os

# 1. Define a simple dictionary to simulate a user database
# Passwords will be stored in plain text to demonstrate insecurity.
users_db = {}

def register_insecure(username, password):
    """
    Registers a new user, storing the password in plain text.
    """
    print(f"\nAttempting to register user: '{username}' with password:
'{password}'")
    if username in users_db:
        print(f"Registration failed: Username '{username}' already
exists.")
        return False
    else:
        users_db[username] = password # Storing plain text password
```

```
        print(f"User '{username}' registered successfully.")
        return True

def login_insecure(username, password):
    """
    Attempts to log in a user by comparing the provided password directly
    with the stored plain-text password.
    """
    print(f"\nAttempting to login user: '{username}' with password:
'{password}'")
    if username not in users_db:
        print(f"Login failed: User '{username}' not found.")
        return False
    else:
        if users_db[username] == password:
            print(f"User '{username}' logged in successfully.")
            return True
        else:
            print(f"Login failed: Incorrect password for user
'{username}'.")
            return False

# --- Test Cases --- (as per instructions)
print("\n--- Demonstrating Insecure Login System ---")

# 1. Register a new user
register_insecure("john_doe", "mysecretpass")

# 2. Register another user
register_insecure("jane_smith", "password123")

# 3. Attempt to register an existing user
register_insecure("john_doe", "newpass")

# 4. Log in with correct credentials
login_insecure("john_doe", "mysecretpass")

# 5. Log in with incorrect credentials
login_insecure("jane_smith", "wrongpass")
```

```
# 6. Attempt to log in with a non-existent user
login_insecure("peter_pan", "neverland")

# 7. Log in with correct credentials for the second user
login_insecure("jane_smith", "password123")

# --- Display users_db to show plain-text passwords ---
print("\n--- Current User Database Content (showing plain-text passwords)
---")
print(users_db)
```

Output:

```
--- Demonstrating Insecure Login System ---

Attempting to register user: 'john_doe' with password: 'mysecretpass'
User 'john_doe' registered successfully.

Attempting to register user: 'jane_smith' with password: 'password123'
User 'jane_smith' registered successfully.

Attempting to register user: 'john_doe' with password: 'newpass'
Registration failed: Username 'john_doe' already exists.

Attempting to login user: 'john_doe' with password: 'mysecretpass'
User 'john_doe' logged in successfully.

Attempting to login user: 'jane_smith' with password: 'wrongpass'
Login failed: Incorrect password for user 'jane_smith'.

Attempting to login user: 'peter_pan' with password: 'neverland'
Login failed: User 'peter_pan' not found.

Attempting to login user: 'jane_smith' with password: 'password123'
User 'jane_smith' logged in successfully.

--- Current User Database Content (showing plain-text passwords) ---
{'john_doe': 'mysecretpass', 'jane_smith':
```

TASK 05

```
import datetime

def log_activity_insecure(username, ip_address, activity_description):
    """
    Logs user activity, including username, IP address, and timestamp,
    without explicit privacy considerations (e.g., logging to console).
    """
    timestamp = datetime.datetime.now()
    log_string = f"[{timestamp}] User: {username} | IP: {ip_address} |
Activity: {activity_description}"
    print(log_string)

# --- Demonstration of Insecure Activity Logging ---
print("\n--- Demonstrating Insecure Activity Logging ---")
log_activity_insecure("alice", "192.168.1.100", "logged in")
log_activity_insecure("bob", "10.0.0.5", "accessed profile")
log_activity_insecure("alice", "192.168.1.100", "downloaded report")
log_activity_insecure("charlie", "172.16.20.30", "failed login attempt")
log_activity_insecure("bob", "10.0.0.5", "updated settings")
```

Output:

```
--- Demonstrating Insecure Activity Logging ---
[2026-01-30 06:40:19.765395] User: alice | IP: 192.168.1.100 | Activity:
logged in
[2026-01-30 06:40:19.765502] User: bob | IP: 10.0.0.5 | Activity: accessed
profile
[2026-01-30 06:40:19.765594] User: alice | IP: 192.168.1.100 | Activity:
downloaded report
[2026-01-30 06:40:19.765689] User: charlie | IP: 172.16.20.30 | Activity:
failed login attempt
[2026-01-30 06:40:19.765766] User: bob | IP: 10.0.0.5 | Activity: updated
settings
```

```
import datetime
import hashlib
```

```
def log_activity_private(username, ip_address, activity_description):
    """
    Logs user activity with privacy-aware techniques: hashing username and masking IP address.

    Args:
        username (str): The username of the activity.
        ip_address (str): The IP address of the user.
        activity_description (str): Description of the activity.
    """
    timestamp = datetime.datetime.now()

    # 1. Anonymize username by hashing
    hashed_username = hashlib.sha256(username.encode('utf-8')).hexdigest()

    # 2. Mask IP address
    # For IPv4, mask the last octet. For IPv6, a more general masking or hashing might be needed.
    # A simple approach for demonstration: split by '.' and replace the last part
    masked_ip = "N/A"
    if '.' in ip_address: # Likely IPv4
        parts = ip_address.split('.')
        if len(parts) == 4:
            masked_ip = f"{parts[0]}.{parts[1]}.{parts[2]}.XXX"
        else:
            masked_ip = "<masked_ipv4_format_error>"
    elif ':' in ip_address: # Likely IPv6 (or a malformed IP)
        # For simplicity, hash the IPv6 address if it's too complex to mask parts
        masked_ip = hashlib.sha256(ip_address.encode('utf-8')).hexdigest()[:12] + "..."
```

```
    else:
        masked_ip = "<unknown_ip_format>"

    # 3. Construct the privacy-aware log string
    log_string = (
        f"[{timestamp}] "
        f"UserHash: {hashed_username[:10]}... | " # Truncate hash for
display
        f"Masked IP: {masked_ip} | "
        f"Activity: {activity_description}"
    )
    print(log_string)

# --- Demonstration of Privacy-Aware Activity Logging ---
print("\n--- Demonstrating Privacy-Aware Activity Logging ---")
log_activity_private("alice", "192.168.1.100", "logged in")
log_activity_private("bob", "10.0.0.5", "accessed profile")
log_activity_private("alice", "192.168.1.101", "downloaded report") #
Alice from a different IP
log_activity_private("charlie", "2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

Output:

```
--- Demonstrating Privacy-Aware Activity Logging ---
[2026-01-30 06:41:02.142562] UserHash: 2bd806c97f... | Masked IP:
192.168.1.XXX | Activity: logged in
[2026-01-30 06:41:02.142742] UserHash: 81b637d8fc... | Masked IP:
10.0.0.XXX | Activity: accessed profile
[2026-01-30 06:41:02.142845] UserHash: 2bd806c97f... | Masked IP:
192.168.1.XXX | Activity: downloaded report
[2026-01-30 06:41:02.142942] UserHash: b9dd960c17... | Masked IP:
50467266fc41... | Activity: failed login attempt
[2026-01-30 06:41:02.143032] UserHash: 81b637d8fc... | Masked IP:
10.0.0.XXX | Activity: updated settings
[2026-01-30 06:41:02.143121] UserHash: 85262adf74... | Masked IP:
172.31.255.XXX | Activity: purchased item
```