SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2023-24

B.Sc. CS 6th Sem

Python Programming Lab Manual (UCS23601J)

Laboratory 1: Display System Information using `pywhois`

Aim

To write a Python program that utilizes the `pywhois` library to retrieve and display system information for a given domain.

Procedure

1. **Install `pywhois`:** If you haven't already, install the `pywhois` library using pip:
2. `pip install python-whois`

3. **Import the library:** In your Python script, import the `whois` module.
4. **Define the target domain:** Choose a domain name (e.g., "google.com") for which you want to retrieve information.
5. **Perform the WHOIS query:** Use `whois.whois()` function with the domain name as an argument.
6. **Print the results:** Display the retrieved information, which will be an object containing various details about the domain.

Source Code

```
# lab1_whois.py

import whois

def get_domain_info(domain_name):
    """
    Retrieves and prints WHOIS information for a given domain.
    """
    try:
        domain_info = whois.whois(domain_name)
        print(f"--- WHOIS Information for {domain_name} ---")
        for key, value in domain_info.items():
            if isinstance(value, list):
                print(f"{key.replace('_', ' ').title()}:")
                for item in value:
                    print(f"  - {item}")
            else:
                print(f"{key.replace('_', ' ').title()}: {value}")
        print("----------------------------------------")
    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    # Example usage:
```

```
    target_domain = "google.com" # You can change this domain
    get_domain_info(target_domain)

    print("\n--- Another example ---")
    target_domain_2 = "python.org"
    get_domain_info(target_domain_2)
```

Input

The domain name is provided directly within the source code (e.g., `target_domain = "google.com"`). No external input is required for this program.

Expected Output

The output will vary based on the domain queried and the current WHOIS database information. Below is a *sample* output structure.

```
--- WHOIS Information for google.com ---
Domain Name: GOOGLE.COM
Registry Domain Id: 2138514_DOMAIN_COM-VRSN
Registrar Whois Server: whois.markmonitor.com
Registrar Url: http://www.markmonitor.com
Updated Date: 2024-02-21T17:00:27Z
Creation Date: 1997-09-15T04:00:00Z
Registrar Registration Expiration Date: 2028-09-14T04:00:00Z
Registrar: MarkMonitor Inc.
Registrar Iana Id: 292
Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
Registrar Abuse Contact Phone: +1.2083895740
Domain Status: clientDeleteProhibited
https://icann.org/epp#clientDeleteProhibited
Domain Status: clientTransferProhibited
https://icann.org/epp#clientTransferProhibited
Domain Status: clientUpdateProhibited
https://icann.org/epp#clientUpdateProhibited
Domain Status: serverDeleteProhibited
https://icann.org/epp#serverDeleteProhibited
Domain Status: serverTransferProhibited
https://icann.org/epp#serverTransferProhibited
Domain Status: serverUpdateProhibited
https://icann.org/epp#serverUpdateProhibited
Name Server: NS1.GOOGLE.COM
Name Server: NS2.GOOGLE.COM
Name Server: NS3.GOOGLE.COM
Name Server: NS4.GOOGLE.COM
Dnssec: unsigned
Url of Whois Database: whois.markmonitor.com
----------------------------------------

--- Another example ---
--- WHOIS Information for python.org ---
Domain Name: PYTHON.ORG
Registry Domain Id: 2138514_DOMAIN_COM-VRSN
Registrar Whois Server: whois.gandi.net
Registrar Url: http://www.gandi.net
Updated Date: 2024-01-26T17:00:27Z
Creation Date: 1997-09-15T04:00:00Z
Registrar Registration Expiration Date: 2028-09-14T04:00:00Z
Registrar: Gandi SAS
Registrar Iana Id: 292
Registrar Abuse Contact Email: abuse@gandi.net
Registrar Abuse Contact Phone: +33.170377661
```

```
Domain Status: clientDeleteProhibited
https://icann.org/epp#clientDeleteProhibited
Domain Status: clientTransferProhibited
https://icann.org/epp#clientTransferProhibited
Domain Status: clientUpdateProhibited
https://icann.org/epp#clientUpdateProhibited
Name Server: NS1.PYTHON.ORG
Name Server: NS2.PYTHON.ORG
Name Server: NS3.PYTHON.ORG
Dnssec: signedDelegation
Url of Whois Database: whois.gandi.net
--------------------------------------------
```

Laboratory 2: The Magic 8 Ball

Aim

To create a Python program that simulates a Magic 8 Ball, providing random answers to user's yes/no questions.

Procedure

1. **Import `random`:** Import the `random` module to choose answers randomly.
2. **Define answers:** Create a list of possible Magic 8 Ball answers (e.g., "It is certain.", "Without a doubt.", "Ask again later.", "Don't count on it.").
3. **Get user input:** Prompt the user to ask a yes/no question.
4. **Choose a random answer:** Use `random.choice()` to select an answer from the list.
5. **Display the answer:** Print the chosen answer to the user.
6. **Loop for multiple questions (optional):** Implement a loop to allow the user to ask multiple questions until they choose to quit.

Source Code

```
# lab2_magic_8_ball.py

import random

def magic_8_ball():
    """
    Simulates a Magic 8 Ball that gives random answers to questions.
    """
    answers = [
        "It is certain.",
        "It is decidedly so.",
        "Without a doubt.",
        "Yes, definitely.",
        "You may rely on it.",
        "As I see it, yes.",
        "Most likely.",
        "Outlook good.",
        "Yes.",
        "Signs point to yes.",
        "Reply hazy, try again.",
        "Ask again later.",
        "Better not tell you now.",
        "Cannot predict now.",
        "Concentrate and ask again.",
        "Don't count on it.",
        "My reply is no.",
        "My sources say no.",
        "Outlook not so good.",
        "Very doubtful."
    ]

    print("Welcome to the Magic 8 Ball!")
    print("Ask me a yes/no question, or type 'quit' to exit.")

    while True:
        question = input("\nWhat is your question? ")
        if question.lower() == 'quit':
            print("Goodbye!")
            break
        elif not question.strip().endswith('?'):
```

```
            print("Please ask a yes/no question ending with a question
mark.")
            continue
        else:
            print(random.choice(answers))

if __name__ == "__main__":
    magic_8_ball()
```

Input

The user will be prompted to enter a yes/no question. Example Input:

```
Will I get a good grade on this exam?
```

Or to quit:

```
quit
```

Expected Output

The output will be one of the predefined answers, chosen randomly.

Example Output 1:

```
Welcome to the Magic 8 Ball!
Ask me a yes/no question, or type 'quit' to exit.

What is your question? Will I get a good grade on this exam?
Most likely.

What is your question? Is today a good day?
Yes, definitely.

What is your question? quit
Goodbye!
```

Example Output 2 (if question doesn't end with '?'):

```
Welcome to the Magic 8 Ball!
Ask me a yes/no question, or type 'quit' to exit.

What is your question? Is this correct
Please ask a yes/no question ending with a question mark.

What is your question? Is this correct?
It is certain.
```

Laboratory 3: Check whether a number is prime or not, Python Program to Generate a Random Number

Aim

To write two separate Python programs:

1. One to determine if a given number is prime.
2. Another to generate a random number within a specified range.

Procedure (Part 1: Prime Number Checker)

1. **Get user input:** Prompt the user to enter a positive integer.
2. **Handle special cases:** Numbers less than or equal to 1 are not prime.
3. **Check divisibility:** Iterate from 2 up to the square root of the number. If the number is divisible by any integer in this range, it's not prime.
4. **Determine primality:** If no divisors are found, the number is prime.

Source Code (Part 1: Prime Number Checker)

```
# lab3_prime_checker.py

def is_prime(num):
    """
    Checks if a given number is prime.
    """
    if num <= 1:
        return False # Numbers less than or equal to 1 are not prime
    if num <= 3:
        return True  # 2 and 3 are prime
    if num % 2 == 0 or num % 3 == 0:
        return False # Multiples of 2 or 3 are not prime

    # Check for prime numbers greater than 3
    # All primes greater than 3 can be written in the form 6k ± 1
    i = 5
    while i * i <= num:
        if num % i == 0 or num % (i + 2) == 0:
            return False
        i += 6
    return True

if __name__ == "__main__":
    print("--- Prime Number Checker ---")
    try:
        number_to_check = int(input("Enter an integer to check for primality:
"))
        if is_prime(number_to_check):
            print(f"{number_to_check} is a prime number.")
        else:
            print(f"{number_to_check} is not a prime number.")
    except ValueError:
        print("Invalid input. Please enter an integer.")
```

Input (Part 1: Prime Number Checker)

An integer entered by the user.

Example Input:

```
Enter an integer to check for primality: 17
```

Or:

```
Enter an integer to check for primality: 10
```

Expected Output (Part 1: Prime Number Checker)

Example Output 1:

```
--- Prime Number Checker ---
Enter an integer to check for primality: 17
17 is a prime number.
```

Example Output 2:

```
--- Prime Number Checker ---
Enter an integer to check for primality: 10
10 is not a prime number.
```

Procedure (Part 2: Random Number Generator)

1. **Import `random`:** Import the `random` module.
2. **Get range input:** Prompt the user to enter the lower and upper bounds for the random number.
3. **Generate random integer:** Use `random.randint()` to generate a random integer within the specified inclusive range.
4. **Display the random number:** Print the generated random number.

Source Code (Part 2: Random Number Generator)

```
# lab3_random_number_generator.py

import random

def generate_random_number(lower_bound, upper_bound):
    """
    Generates and prints a random integer within a specified range.
    """
    if lower_bound > upper_bound:
        print("Error: Lower bound cannot be greater than upper bound.")
        return

    random_num = random.randint(lower_bound, upper_bound)
    print(f"Generated random number between {lower_bound} and {upper_bound}:
{random_num}")

if __name__ == "__main__":
    print("\n--- Random Number Generator ---")
    try:
        min_val = int(input("Enter the lower bound: "))
        max_val = int(input("Enter the upper bound: "))
        generate_random_number(min_val, max_val)
    except ValueError:
        print("Invalid input. Please enter integers for bounds.")
```

Input (Part 2: Random Number Generator)

Two integers representing the lower and upper bounds of the range.

Example Input:

```
Enter the lower bound: 1
Enter the upper bound: 100
```

Expected Output (Part 2: Random Number Generator)

The output will be a random integer within the specified range.

Example Output:

```
--- Random Number Generator ---
Enter the lower bound: 1
Enter the upper bound: 100
Generated random number between 1 and 100: 45
```

(The number 45 is just an example; it will be different each time.)

Laboratory 4: Make a simple calculator

Aim

To create a Python program that functions as a simple calculator, performing basic arithmetic operations (addition, subtraction, multiplication, division) based on user input.

Procedure

1. **Define functions for operations:** Create separate functions for addition, subtraction, multiplication, and division.
2. **Display operation choices:** Present a menu to the user for selecting an operation.
3. **Get user input:**
    - Prompt the user to choose an operation.
    - Prompt the user to enter two numbers.
4. **Perform calculation:** Based on the chosen operation, call the corresponding function with the two numbers.
5. **Handle division by zero:** Implement error handling for division by zero.
6. **Display result:** Print the result of the calculation.
7. **Loop for multiple calculations (optional):** Allow the user to perform multiple calculations until they choose to quit.

Source Code

```
# lab4_simple_calculator.py

def add(x, y):
    """Adds two numbers."""
    return x + y

def subtract(x, y):
    """Subtracts two numbers."""
    return x - y

def multiply(x, y):
    """Multiplies two numbers."""
    return x * y

def divide(x, y):
    """Divides two numbers, handles division by zero."""
    if y == 0:
        return "Error! Division by zero."
    return x / y

def simple_calculator():
    """
    Runs a simple calculator program.
    """
    print("--- Simple Calculator ---")
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    print("Enter 'quit' to exit")

    while True:
        choice = input("Enter choice(1/2/3/4/quit): ").lower()

        if choice == 'quit':
            print("Exiting calculator. Goodbye!")
```

```
            break

        if choice in ('1', '2', '3', '4'):
            try:
                num1 = float(input("Enter first number: "))
                num2 = float(input("Enter second number: "))
            except ValueError:
                print("Invalid input. Please enter numbers.")
                continue

            if choice == '1':
                print(f"{num1} + {num2} = {add(num1, num2)}")
            elif choice == '2':
                print(f"{num1} - {num2} = {subtract(num1, num2)}")
            elif choice == '3':
                print(f"{num1} * {num2} = {multiply(num1, num2)}")
            elif choice == '4':
                result = divide(num1, num2)
                print(f"{num1} / {num2} = {result}")
        else:
            print("Invalid input. Please enter a valid choice (1/2/3/4) or
'quit'.")

if __name__ == "__main__":
    simple_calculator()
```

Input

The user will input their choice of operation (1-4 or 'quit') and two numbers.

Example Input:

```
Enter choice(1/2/3/4/quit): 3
Enter first number: 10
Enter second number: 5
```

Example Input for division by zero:

```
Enter choice(1/2/3/4/quit): 4
Enter first number: 10
Enter second number: 0
```

Expected Output

Example Output 1 (Multiplication):

```
--- Simple Calculator ---
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter 'quit' to exit
Enter choice(1/2/3/4/quit): 3
Enter first number: 10
Enter second number: 5
10.0 * 5.0 = 50.0
Enter choice(1/2/3/4/quit): quit
Exiting calculator. Goodbye!
```

Example Output 2 (Division by Zero):

```
--- Simple Calculator ---
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter 'quit' to exit
Enter choice(1/2/3/4/quit): 4
Enter first number: 10
Enter second number: 0
10.0 / 0.0 = Error! Division by zero.
Enter choice(1/2/3/4/quit): quit
Exiting calculator. Goodbye!
```

Laboratory 5: Find the Factorial of a Number, Python Program to Convert Decimal to Binary, Octal and Hexadecimal

Aim

To write two separate Python programs:

1. One to calculate the factorial of a non-negative integer.
2. Another to convert a decimal number into its binary, octal, and hexadecimal representations.

Procedure (Part 1: Factorial of a Number)

1. **Get user input:** Prompt the user to enter a non-negative integer.
2. **Handle negative input:** Factorial is not defined for negative numbers.
3. **Handle base cases:** Factorial of 0 is 1.
4. **Calculate factorial:** Use a loop or recursion to multiply all integers from 1 up to the given number.
5. **Display result:** Print the calculated factorial.

Source Code (Part 1: Factorial of a Number)

```
# lab5_factorial.py

def factorial(n):
    """
    Calculates the factorial of a non-negative integer.
    """
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0:
        return 1
    else:
        fact = 1
        for i in range(1, n + 1):
            fact *= i
        return fact

if __name__ == "__main__":
    print("--- Factorial Calculator ---")
    try:
        num = int(input("Enter a non-negative integer: "))
        result = factorial(num)
        print(f"The factorial of {num} is: {result}")
    except ValueError:
        print("Invalid input. Please enter a non-negative integer.")
```

Input (Part 1: Factorial of a Number)

A non-negative integer entered by the user.

Example Input:

```
Enter a non-negative integer: 5
```

Or:

```
Enter a non-negative integer: -3
```

Expected Output (Part 1: Factorial of a Number)

Example Output 1:

```
--- Factorial Calculator ---
Enter a non-negative integer: 5
The factorial of 5 is: 120
```

Example Output 2:

```
--- Factorial Calculator ---
Enter a non-negative integer: -3
The factorial of -3 is: Factorial is not defined for negative numbers.
```

Procedure (Part 2: Decimal to Binary, Octal, Hexadecimal Converter)

1. **Get user input:** Prompt the user to enter a decimal integer.
2. **Convert to binary:** Use the built-in `bin()` function.
3. **Convert to octal:** Use the built-in `oct()` function.
4. **Convert to hexadecimal:** Use the built-in `hex()` function.
5. **Display results:** Print all three converted values. Note that `bin()`, `oct()`, and `hex()` return strings with prefixes (`0b`, `0o`, `0x`), which can be removed if desired using slicing.

Source Code (Part 2: Decimal to Binary, Octal, Hexadecimal Converter)

```
# lab5_decimal_converter.py

def convert_decimal(decimal_num):
    """
    Converts a decimal number to binary, octal, and hexadecimal.
    """
    print(f"\n--- Conversions for Decimal: {decimal_num} ---")
    print(f"Binary: {bin(decimal_num)}")
    print(f"Octal: {oct(decimal_num)}")
    print(f"Hexadecimal: {hex(decimal_num)}")
    print("-----------------------------------------")

if __name__ == "__main__":
    print("--- Decimal Number Converter ---")
    try:
        dec_num = int(input("Enter a decimal integer: "))
        convert_decimal(dec_num)
    except ValueError:
        print("Invalid input. Please enter an integer.")
```

Input (Part 2: Decimal to Binary, Octal, Hexadecimal Converter)

A decimal integer entered by the user.

Example Input:

```
Enter a decimal integer: 255
```

Expected Output (Part 2: Decimal to Binary, Octal, Hexadecimal Converter)

Example Output:

```
--- Decimal Number Converter ---
Enter a decimal integer: 255

--- Conversions for Decimal: 255 ---
Binary: 0b11111111
Octal: 0o377
Hexadecimal: 0xff
-----------------------------------------
```

Laboratory 6: Program to read and write text and numbers

Aim

To write a Python program that demonstrates reading from and writing to text files, handling both string (text) and numerical data.

Procedure

1. **Writing to a file:**
   - Open a file in write mode (`'w'`) or append mode (`'a'`).
   - Write text strings using `file.write()`.
   - Convert numbers to strings before writing them to the file.
   - Close the file using `file.close()` or use a `with` statement for automatic closing.
2. **Reading from a file:**
   - Open the file in read mode (`'r'`).
   - Read the entire content using `file.read()`, or line by line using `file.readline()` or iterating over the file object.
   - If reading numbers, convert the string back to an integer or float using `int()` or `float()`.
   - Close the file.

Source Code

```
# lab6_file_io.py

def write_data_to_file(filename="my_data.txt"):
    """
    Writes text and numbers to a specified file.
    """
    print(f"--- Writing data to '{filename}' ---")
    try:
        with open(filename, 'w') as file:
            file.write("Hello, this is a line of text.\n")
            file.write("Python file I/O demonstration.\n")

            # Writing numbers
            num1 = 123
            num2 = 45.67
            file.write(f"First number: {num1}\n")
            file.write(f"Second number: {num2}\n")
            file.write("End of file.\n")
        print("Data successfully written.")
    except IOError as e:
        print(f"Error writing to file: {e}")

def read_data_from_file(filename="my_data.txt"):
    """
    Reads text and numbers from a specified file.
    """
    print(f"\n--- Reading data from '{filename}' ---")
    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
            for i, line in enumerate(lines):
                print(f"Line {i+1}: {line.strip()}") # .strip() removes
newline characters

                # Example of parsing numbers if they are in a known format
                if "First number:" in line:
```

```
                    try:
                        number_str = line.split(":")[1].strip()
                        number = int(number_str)
                        print(f"  (Parsed integer: {number})")
                    except (ValueError, IndexError):
                        pass # Not a number or format not as expected
                elif "Second number:" in line:
                    try:
                        number_str = line.split(":")[1].strip()
                        number = float(number_str)
                        print(f"  (Parsed float: {number})")
                    except (ValueError, IndexError):
                        pass
        print("Data successfully read.")
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
    except IOError as e:
        print(f"Error reading from file: {e}")

if __name__ == "__main__":
    data_file = "sample_data.txt"
    write_data_to_file(data_file)
    read_data_from_file(data_file)

    # Demonstrate appending to a file
    print("\n--- Appending more data ---")
    with open(data_file, 'a') as file:
        file.write("This line was appended.\n")
        file.write("Another appended number: 99.9\n")
    print("More data appended.")
    read_data_from_file(data_file)
```

Input

No direct user input is required for this program. The data is hardcoded within the `write_data_to_file` function.

Expected Output

```
--- Writing data to 'sample_data.txt' ---
Data successfully written.

--- Reading data from 'sample_data.txt' ---
Line 1: Hello, this is a line of text.
Line 2: Python file I/O demonstration.
Line 3: First number: 123
  (Parsed integer: 123)
Line 4: Second number: 45.67
  (Parsed float: 45.67)
Line 5: End of file.
Data successfully read.

--- Appending more data ---
More data appended.

--- Reading data from 'sample_data.txt' ---
Line 1: Hello, this is a line of text.
Line 2: Python file I/O demonstration.
Line 3: First number: 123
  (Parsed integer: 123)
Line 4: Second number: 45.67
  (Parsed float: 45.67)
Line 5: End of file.
Line 6: This line was appended.
```

```
Line 7: Another appended number: 99.9
  (Parsed float: 99.9)
Data successfully read.
```

Laboratory 7: Program to Transpose a Matrix, Program to List Methods for Inserting Elements

Aim

To write two separate Python programs:

1. One to transpose a given matrix (represented as a list of lists).
2. Another to demonstrate various methods for inserting elements into different Python data structures (lists, tuples, sets, dictionaries).

Procedure (Part 1: Transpose a Matrix)

1. **Define the matrix:** Represent the matrix as a list of lists.
2. **Determine dimensions:** Get the number of rows and columns of the original matrix.
3. **Create a new matrix for transpose:** Initialize a new matrix with dimensions (columns x rows) filled with zeros or a placeholder.
4. **Populate transposed matrix:** Iterate through the original matrix, swapping rows and columns (i.e., `transposed_matrix[j][i] = original_matrix[i][j]`).
5. **Display matrices:** Print both the original and transposed matrices.

Source Code (Part 1: Transpose a Matrix)

```
# lab7_matrix_transpose.py

def transpose_matrix(matrix):
    """
    Transposes a given matrix (list of lists).
    """
    if not matrix:
        return []

    rows = len(matrix)
    cols = len(matrix[0])

    # Create a new matrix with swapped dimensions, initialized with zeros
    transposed = [[0 for _ in range(rows)] for _ in range(cols)]

    # Populate the transposed matrix
    for i in range(rows):
        for j in range(cols):
            transposed[j][i] = matrix[i][j]
    return transposed

def print_matrix(matrix, name="Matrix"):
    """Helper function to print a matrix."""
    print(f"\n--- {name} ---")
    for row in matrix:
        print(row)
    print("-----------------")

if __name__ == "__main__":
    original_matrix = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]

    print_matrix(original_matrix, "Original Matrix")
```

```
    transposed_mat = transpose_matrix(original_matrix)
    print_matrix(transposed_mat, "Transposed Matrix")

    # Example with a non-square matrix
    print("\n--- Non-Square Matrix Example ---")
    non_square_matrix = [
        [10, 20],
        [30, 40],
        [50, 60]
    ]
    print_matrix(non_square_matrix, "Original Non-Square Matrix")
    transposed_non_square = transpose_matrix(non_square_matrix)
    print_matrix(transposed_non_square, "Transposed Non-Square Matrix")
```

Input (Part 1: Transpose a Matrix)

The matrix is hardcoded within the source code. No user input is required.

Example Matrix:

```
original_matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

Expected Output (Part 1: Transpose a Matrix)

```
--- Original Matrix ---
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
-----------------

--- Transposed Matrix ---
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
-----------------

--- Non-Square Matrix Example ---

--- Original Non-Square Matrix ---
[10, 20]
[30, 40]
[50, 60]
-----------------

--- Transposed Non-Square Matrix ---
[10, 30, 50]
[20, 40, 60]
-----------------
```

Procedure (Part 2: List Methods for Inserting Elements)

1. **Initialize data structures:** Create example lists, tuples, sets, and dictionaries.
2. **Lists:** Demonstrate `append()`, `insert()`, and `extend()`.
3. **Tuples:** Explain that tuples are immutable and cannot have elements inserted directly. Show how to create a new tuple with inserted elements.
4. **Sets:** Demonstrate `add()` and `update()`.

5. **Dictionaries:** Demonstrate adding new key-value pairs and using `update()`.
6. **Print results:** Display the data structures after each insertion operation.

Source Code (Part 2: List Methods for Inserting Elements)

```
# lab7_insert_methods.py

def demonstrate_insert_methods():
    """
    Demonstrates various methods for inserting elements into Python data
structures.
    """
    print("--- Demonstrating Insertion Methods ---")

    # 1. Lists
    print("\n1. Lists:")
    my_list = [10, 20, 30]
    print(f"  Original List: {my_list}")

    # append(): Adds an element to the end of the list
    my_list.append(40)
    print(f"  After append(40): {my_list}")

    # insert(): Inserts an element at a specified index
    my_list.insert(1, 15) # Insert 15 at index 1
    print(f"  After insert(1, 15): {my_list}")

    # extend(): Extends the list by appending all the items from an iterable
    another_list = [50, 60]
    my_list.extend(another_list)
    print(f"  After extend([50, 60]): {my_list}")

    # 2. Tuples (Immutable)
    print("\n2. Tuples (Immutable - Cannot insert directly):")
    my_tuple = (1, 2, 3)
    print(f"  Original Tuple: {my_tuple}")
    print("  Tuples are immutable. To 'insert' an element, you create a new
tuple.")

    # Example of creating a new tuple with an "inserted" element
    new_tuple = my_tuple[:1] + (99,) + my_tuple[1:]
    print(f"  New Tuple (insert 99 at index 1): {new_tuple}")

    # 3. Sets
    print("\n3. Sets:")
    my_set = {1, 2, 3}
    print(f"  Original Set: {my_set}")

    # add(): Adds a single element to the set
    my_set.add(4)
    print(f"  After add(4): {my_set}")

    # update(): Adds elements from an iterable (like a list or another set)
    my_set.update([5, 6])
    print(f"  After update([5, 6]): {my_set}")
    my_set.add(3) # Adding an existing element has no effect
    print(f"  After add(3) (no change): {my_set}")

    # 4. Dictionaries
    print("\n4. Dictionaries:")
    my_dict = {'a': 1, 'b': 2}
    print(f"  Original Dictionary: {my_dict}")

    # Adding a new key-value pair directly
    my_dict['c'] = 3
    print(f"  After my_dict['c'] = 3: {my_dict}")
```

```
    # update(): Adds key-value pairs from another dictionary or from keyword
arguments
    my_dict.update({'d': 4, 'e': 5})
    print(f"  After update({{'d': 4, 'e': 5}}): {my_dict}")

    my_dict.update(f=6, g=7)
    print(f"  After update(f=6, g=7): {my_dict}")

    print("\n-----------------------------------")

if __name__ == "__main__":
    demonstrate_insert_methods()
```

Input (Part 2: List Methods for Inserting Elements)

No user input is required. All data structures and operations are hardcoded.

Expected Output (Part 2: List Methods for Inserting Elements)

```
--- Demonstrating Insertion Methods ---

1. Lists:
  Original List: [10, 20, 30]
  After append(40): [10, 20, 30, 40]
  After insert(1, 15): [10, 15, 20, 30, 40]
  After extend([50, 60]): [10, 15, 20, 30, 40, 50, 60]

2. Tuples (Immutable - Cannot insert directly):
  Original Tuple: (1, 2, 3)
  Tuples are immutable. To 'insert' an element, you create a new tuple.
  New Tuple (insert 99 at index 1): (1, 99, 2, 3)

3. Sets:
  Original Set: {1, 2, 3}
  After add(4): {1, 2, 3, 4}
  After update([5, 6]): {1, 2, 3, 4, 5, 6}
  After add(3) (no change): {1, 2, 3, 4, 5, 6}

4. Dictionaries:
  Original Dictionary: {'a': 1, 'b': 2}
  After my_dict['c'] = 3: {'a': 1, 'b': 2, 'c': 3}
  After update({'d': 4, 'e': 5}): {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
  After update(f=6, g=7): {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6,
'g': 7}

-----------------------------------
```

Laboratory 8: Using a List to Find the Median of a Set of Numbers, Program using sorting and searching

Aim

To write two separate Python programs:

1. One to calculate the median of a list of numbers.
2. Another to demonstrate sorting and searching algorithms on a list of numbers.

Procedure (Part 1: Find the Median of a Set of Numbers)

1. **Get input list:** Define a list of numbers.
2. **Sort the list:** Sort the list in ascending order. This is crucial for finding the median.
3. **Determine list length:** Get the number of elements in the sorted list.
4. **Check for odd/even length:**
    - If the length is odd, the median is the middle element.
    - If the length is even, the median is the average of the two middle elements.
5. **Display the median:** Print the calculated median.

Source Code (Part 1: Find the Median of a Set of Numbers)

```
# lab8_median.py

def find_median(numbers):
    """
    Finds the median of a list of numbers.
    """
    if not numbers:
        return None # Median is undefined for an empty list

    # Sort the list
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)

    if n % 2 == 1:
        # Odd number of elements, median is the middle element
        median = sorted_numbers[n // 2]
    else:
        # Even number of elements, median is the average of the two middle
elements
        mid1 = sorted_numbers[n // 2 - 1]
        mid2 = sorted_numbers[n // 2]
        median = (mid1 + mid2) / 2

    return median

if __name__ == "__main__":
    print("--- Median Calculator ---")

    list1 = [1, 3, 2, 5, 4] # Odd number of elements
    print(f"List: {list1}")
    print(f"Median: {find_median(list1)}")

    list2 = [10, 20, 30, 40] # Even number of elements
    print(f"\nList: {list2}")
    print(f"Median: {find_median(list2)}")

    list3 = [7] # Single element
    print(f"\nList: {list3}")
```

```
    print(f"Median: {find_median(list3)}")

    list4 = [] # Empty list
    print(f"\nList: {list4}")
    print(f"Median: {find_median(list4)}")
```

Input (Part 1: Find the Median of a Set of Numbers)

The list of numbers is hardcoded within the source code. No user input is required.

Example List:

```
numbers = [1, 3, 2, 5, 4]
```

Expected Output (Part 1: Find the Median of a Set of Numbers)

```
--- Median Calculator ---
List: [1, 3, 2, 5, 4]
Median: 3

List: [10, 20, 30, 40]
Median: 25.0

List: [7]
Median: 7

List: []
Median: None
```

Procedure (Part 2: Program using sorting and searching)

1. **Define a list:** Create an unsorted list of numbers.
2. **Sorting:**
    - Demonstrate `list.sort()` (in-place sort).
    - Demonstrate `sorted()` (returns a new sorted list).
3. **Searching (Linear Search):**
    - Implement a linear search function that iterates through the list to find an element.
    - Return the index if found, otherwise indicate not found.
4. **Searching (Binary Search - requires sorted list):**
    - Implement a binary search function (requires the list to be sorted).
    - Explain the prerequisites for binary search.
    - Return the index if found, otherwise indicate not found.
5. **Display results:** Print the list before and after sorting, and the results of search operations.

Source Code (Part 2: Program using sorting and searching)

```
# lab8_sorting_searching.py

def linear_search(data_list, target):
    """
    Performs a linear search for a target element in a list.
    Returns the index of the target if found, otherwise -1.
    """
    for i in range(len(data_list)):
        if data_list[i] == target:
            return i
    return -1
```

```
def binary_search(data_list, target):
    """
    Performs a binary search for a target element in a SORTED list.
    Returns the index of the target if found, otherwise -1.
    """
    low = 0
    high = len(data_list) - 1

    while low <= high:
        mid = (low + high) // 2
        if data_list[mid] == target:
            return mid
        elif data_list[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

if __name__ == "__main__":
    my_numbers = [64, 34, 25, 12, 22, 11, 90]
    print(f"Original List: {my_numbers}")

    # --- Sorting Demonstrations ---
    print("\n--- Sorting ---")

    # Using list.sort() (in-place)
    list_for_sort = my_numbers[:] # Create a copy to show in-place
modification
    list_for_sort.sort()
    print(f"List after list.sort() (in-place): {list_for_sort}")

    # Using sorted() (returns new list)
    new_sorted_list = sorted(my_numbers)
    print(f"New list after sorted(): {new_sorted_list}")
    print(f"Original list remains unchanged: {my_numbers}") # my_numbers is
still unsorted

    # --- Searching Demonstrations ---
    print("\n--- Searching ---")

    search_list = sorted(my_numbers) # Use a sorted list for binary search
    print(f"List for searching (sorted): {search_list}")

    # Linear Search
    target1 = 22
    index1 = linear_search(search_list, target1)
    if index1 != -1:
        print(f"Linear Search: {target1} found at index {index1}")
    else:
        print(f"Linear Search: {target1} not found")

    target2 = 100
    index2 = linear_search(search_list, target2)
    if index2 != -1:
        print(f"Linear Search: {target2} found at index {index2}")
    else:
        print(f"Linear Search: {target2} not found")

    # Binary Search (requires sorted list)
    print("\nBinary Search (requires sorted list):")
    target3 = 11
    index3 = binary_search(search_list, target3)
    if index3 != -1:
        print(f"Binary Search: {target3} found at index {index3}")
    else:
```

```
        print(f"Binary Search: {target3} not found")

    target4 = 50
    index4 = binary_search(search_list, target4)
    if index4 != -1:
        print(f"Binary Search: {target4} found at index {index4}")
    else:
        print(f"Binary Search: {target4} not found")
```

Input (Part 2: Program using sorting and searching)

The list of numbers and search targets are hardcoded. No user input is required.

Example List:

```
my_numbers = [64, 34, 25, 12, 22, 11, 90]
```

Expected Output (Part 2: Program using sorting and searching)

```
Original List: [64, 34, 25, 12, 22, 11, 90]

--- Sorting ---
List after list.sort() (in-place): [11, 12, 22, 25, 34, 64, 90]
New list after sorted(): [11, 12, 22, 25, 34, 64, 90]
Original list remains unchanged: [64, 34, 25, 12, 22, 11, 90]

--- Searching ---
List for searching (sorted): [11, 12, 22, 25, 34, 64, 90]
Linear Search: 22 found at index 2
Linear Search: 100 not found

Binary Search (requires sorted list):
Binary Search: 11 found at index 0
Binary Search: 50 not found
```

Laboratory 9: Eliza-like Chatbot (Responding with hedges and keyword transformations)

Aim

To create a Python program that simulates a simple conversational agent (chatbot) similar to Eliza, responding to user statements by either providing a random "hedge" or transforming keywords in the user's input.

Procedure

1. **Define hedges:** Create a list of generic, non-committal responses (hedges).
2. **Define keyword transformations:** Create a dictionary mapping common user keywords to their transformed counterparts (e.g., "I" to "you", "my" to "your").
3. **Implement response logic:**
    - Randomly choose between a hedge response or a keyword transformation response.
    - If keyword transformation:
        - Split the user's input into words.
        - Iterate through the words and replace any matching keywords with their transformed versions.
        - Construct a new sentence using a randomly chosen qualifier (e.g., "Why do you say that...") and the transformed user input.
4. **Loop for conversation:** Allow the user to continue the conversation until they choose to quit.

Source Code

```
# lab9_eliza_chatbot.py

import random
import re # For regular expressions to handle word boundaries

def eliza_chatbot():
    """
    Simulates a simple Eliza-like chatbot.
    """
    hedges = [
        "Please tell me more.",
        "Many of my patients tell me the same thing.",
        "Go on.",
        "That's interesting. Can you elaborate?",
        "How does that make you feel?",
        "I see. And what else is on your mind?"
    ]

    # Mapping for keyword transformations (user_word: bot_word)
    # Using regex word boundaries for more accurate replacements
    # Note: Order matters for some transformations (e.g., 'my' before 'me')
    transformations = {
        r'\bI\b': 'you',
        r'\bme\b': 'you',
        r'\bmy\b': 'your',
        r'\bmine\b': 'yours',
        r'\byou\b': 'I',
        r'\byour\b': 'my',
        r'\byours\b': 'mine',
        r'\bam\b': 'are',
        r'\bare\b': 'am',
```

```
        r'\bwas\b': 'were',
        r'\bwere\b': 'was',
        r'\bI\'m\b': 'you are',
        r'\bI was\b': 'you were'
    }

    qualifiers = [
        "Why do you say that ",
        "Can you explain why ",
        "What makes you think that ",
        "Tell me more about why "
    ]

    print("--- Eliza-like Chatbot ---")
    print("Hello. How can I help you today? (Type 'quit' to exit)")

    while True:
        user_input = input("> ").strip()
        if user_input.lower() == 'quit':
            print("Goodbye! It was nice talking to you.")
            break

        # Randomly choose between a hedge or a transformation
        if random.choice([True, False]): # 50% chance for either
            response = random.choice(hedges)
        else:
            # Apply transformations
            transformed_input = user_input
            for old_word_pattern, new_word in transformations.items():
                # Use re.sub for robust word replacement
                transformed_input = re.sub(old_word_pattern, new_word,
transformed_input, flags=re.IGNORECASE)

            # Capitalize the first letter of the transformed input if it
starts with 'i'
            if transformed_input.lower().startswith('i '):
                transformed_input = 'I' + transformed_input[1:]

            # Append a random qualifier
            response = random.choice(qualifiers) + transformed_input + "?"

        print(response)

if __name__ == "__main__":
    eliza_chatbot()
```

Input

The user will type statements or questions.

Example Input:

```
My teacher always plays favorites.
I am feeling sad today.
You never listen to me.
```

Expected Output

The output will vary based on the random choice of response type and the keyword transformations.

Example Output:

```
--- Eliza-like Chatbot ---
Hello. How can I help you today? (Type 'quit' to exit)
> My teacher always plays favorites.
Why do you say that your teacher always plays favorites?
> I am feeling sad today.
Tell me more about why you are feeling sad today?
> You never listen to me.
Please tell me more.
> I was thinking about my future.
What makes you think that you were thinking about your future?
> quit
Goodbye! It was nice talking to you.
```

Laboratory 10: Program using recursive function

Aim

To write a Python program that demonstrates the concept of recursion by implementing a common recursive function, such as calculating the Fibonacci sequence or the factorial of a number.

Procedure

1. **Choose a recursive problem:** Factorial is a good simple example.
2. **Define the base case:** Identify the condition under which the recursion stops (e.g., factorial of 0 or 1).
3. **Define the recursive step:** Express the problem in terms of a smaller instance of itself (e.g., $n! = n \times (n-1)!$).
4. **Implement the function:** Write the Python function incorporating the base case and the recursive step.
5. **Get user input:** Prompt the user for the number.
6. **Call the function and display result:** Print the result.

Source Code

```
# lab10_recursive_function.py

def factorial_recursive(n):
    """
    Calculates the factorial of a non-negative integer using recursion.
    """
    # Base case: Factorial of 0 is 1
    if n == 0:
        return 1
    # Handle negative input (optional, or raise an error)
    elif n < 0:
        return "Factorial is not defined for negative numbers."
    # Recursive step: n! = n * (n-1)!
    else:
        return n * factorial_recursive(n - 1)

def fibonacci_recursive(n):
    """
    Calculates the nth Fibonacci number using recursion.
    F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)
    """
    # Base cases
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    # Recursive step
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

if __name__ == "__main__":
    print("--- Recursive Function Demonstrations ---")

    # --- Factorial Example ---
    print("\n--- Factorial Calculator (Recursive) ---")
    try:
        num_fact = int(input("Enter a non-negative integer for factorial: "))
        result_fact = factorial_recursive(num_fact)
        print(f"The factorial of {num_fact} is: {result_fact}")
```

```
    except ValueError:
        print("Invalid input. Please enter an integer.")

    # --- Fibonacci Example ---
    print("\n--- Fibonacci Sequence Generator (Recursive) ---")
    try:
        num_fib = int(input("Enter a non-negative integer for Fibonacci (n-th
term): "))
        if num_fib < 0:
            print("Fibonacci sequence is defined for non-negative integers.")
        else:
            result_fib = fibonacci_recursive(num_fib)
            print(f"The {num_fib}-th Fibonacci number is: {result_fib}")

            # Print sequence up to n
            print(f"Fibonacci sequence up to {num_fib}:")
            fib_sequence = [fibonacci_recursive(i) for i in range(num_fib +
1)]
            print(fib_sequence)

    except ValueError:
        print("Invalid input. Please enter an integer.")
```

Input

The user will enter a non-negative integer for both factorial and Fibonacci calculations.

Example Input for Factorial:

```
Enter a non-negative integer for factorial: 7
```

Example Input for Fibonacci:

```
Enter a non-negative integer for Fibonacci (n-th term): 8
```

Expected Output

Example Output for Factorial:

```
--- Recursive Function Demonstrations ---

--- Factorial Calculator (Recursive) ---
Enter a non-negative integer for factorial: 7
The factorial of 7 is: 5040
```

Example Output for Fibonacci:

```
--- Fibonacci Sequence Generator (Recursive) ---
Enter a non-negative integer for Fibonacci (n-th term): 8
The 8-th Fibonacci number is: 21
Fibonacci sequence up to 8:
[0, 1, 1, 2, 3, 5, 8, 13, 21]
```

Laboratory 11: Write the code for a mapping that generates a list of the absolute values of the numbers in a list named numbers.

Aim

To write a Python program that demonstrates the concept of mapping (transformation) by generating a new list containing the absolute values of numbers from an existing list. This will be achieved using a loop, list comprehension, and the `map()` function.

Procedure

1. **Define the original list:** Create a list of numbers, including positive and negative values.
2. **Method 1: Using a `for` loop:** Iterate through the original list, apply `abs()` to each number, and append the result to a new list.
3. **Method 2: Using List Comprehension:** Create a new list using a concise list comprehension that applies `abs()` to each element.
4. **Method 3: Using `map()` function:** Use the `map()` function with `abs` as the function and the original list as the iterable. Convert the map object to a list.
5. **Display results:** Print the original list and the new lists generated by each method.

Source Code

```
# lab11_absolute_values_mapping.py

def generate_absolute_values():
    """
    Generates a list of absolute values from an original list using different
mapping methods.
    """
    numbers = [-5, 10, -15, 0, 20, -25]
    print(f"Original list: {numbers}")

    # Method 1: Using a for loop
    absolute_values_loop = []
    for num in numbers:
        absolute_values_loop.append(abs(num))
    print(f"Absolute values (using for loop): {absolute_values_loop}")

    # Method 2: Using List Comprehension
    absolute_values_comprehension = [abs(num) for num in numbers]
    print(f"Absolute values (using list comprehension):
{absolute_values_comprehension}")

    # Method 3: Using map() function
    # map() returns a map object, so convert it to a list
    absolute_values_map = list(map(abs, numbers))
    print(f"Absolute values (using map() function): {absolute_values_map}")

if __name__ == "__main__":
    print("--- Mapping: Generating Absolute Values ---")
    generate_absolute_values()
```

Input

The list of numbers is hardcoded within the source code. No user input is required.

Example List:

```
numbers = [-5, 10, -15, 0, 20, -25]
```

Expected Output

```
--- Mapping: Generating Absolute Values ---
Original list: [-5, 10, -15, 0, 20, -25]
Absolute values (using for loop): [5, 10, 15, 0, 20, 25]
Absolute values (using list comprehension): [5, 10, 15, 0, 20, 25]
Absolute values (using map() function): [5, 10, 15, 0, 20, 25]
```

Laboratory 12: Write the code for a filtering that generates a list of the positive numbers in a list named numbers. You should use a lambda to create the auxiliary function

Aim

To write a Python program that demonstrates the concept of filtering by generating a new list containing only the positive numbers from an existing list. This will be achieved using a loop, list comprehension, and the `filter()` function with a `lambda` expression.

Procedure

1. **Define the original list:** Create a list of numbers, including positive, negative, and zero values.
2. **Method 1: Using a `for` loop:** Iterate through the original list, check if each number is positive, and append it to a new list if it is.
3. **Method 2: Using List Comprehension:** Create a new list using a concise list comprehension with a conditional filter.
4. **Method 3: Using `filter()` function with `lambda`:** Use the `filter()` function with a `lambda` expression (e.g., `lambda x: x > 0`) as the filtering function and the original list as the iterable. Convert the filter object to a list.
5. **Display results:** Print the original list and the new lists generated by each method.

Source Code

```
# lab12_positive_numbers_filtering.py

def filter_positive_numbers():
    """
    Filters positive numbers from an original list using different methods,
    including filter() with a lambda function.
    """
    numbers = [-10, 5, 0, -3, 12, -7, 8, 0]
    print(f"Original list: {numbers}")

    # Method 1: Using a for loop
    positive_numbers_loop = []
    for num in numbers:
        if num > 0:
            positive_numbers_loop.append(num)
    print(f"Positive numbers (using for loop): {positive_numbers_loop}")

    # Method 2: Using List Comprehension
    positive_numbers_comprehension = [num for num in numbers if num > 0]
    print(f"Positive numbers (using list comprehension):
{positive_numbers_comprehension}")

    # Method 3: Using filter() function with a lambda
    # The lambda function `lambda x: x > 0` returns True for positive numbers
    positive_numbers_filter_lambda = list(filter(lambda x: x > 0, numbers))
    print(f"Positive numbers (using filter() with lambda):
{positive_numbers_filter_lambda}")

if __name__ == "__main__":
    print("--- Filtering: Generating Positive Numbers ---")
    filter_positive_numbers()
```

Input

The list of numbers is hardcoded within the source code. No user input is required.

Example List:

```
numbers = [-10, 5, 0, -3, 12, -7, 8, 0]
```

Expected Output

```
--- Filtering: Generating Positive Numbers ---
Original list: [-10, 5, 0, -3, 12, -7, 8, 0]
Positive numbers (using for loop): [5, 12, 8]
Positive numbers (using list comprehension): [5, 12, 8]
Positive numbers (using filter() with lambda): [5, 12, 8]
```

Laboratory 13: Program using classes and methods

Aim

To write a Python program that demonstrates Object-Oriented Programming (OOP) concepts by defining a class with attributes (data) and methods (functions) to represent a real-world entity.

Procedure

1. **Define a Class:** Create a class (e.g., `Car`, `Book`, `Student`).
2. **Define `__init__` method (Constructor):** Initialize the attributes of the class when an object is created.
3. **Define Instance Methods:** Create methods that operate on the attributes of the class instance. These methods should take `self` as the first parameter.
4. **Create Objects (Instances):** Create multiple objects of the defined class.
5. **Access Attributes and Call Methods:** Access the attributes of the objects and call their methods to demonstrate functionality.

Source Code

```
# lab13_classes_methods.py

class Book:
    """
    Represents a book with a title, author, ISBN, and availability status.
    """
    def __init__(self, title, author, isbn, available=True):
        """
        Initializes a new Book object.

        Args:
            title (str): The title of the book.
            author (str): The author of the book.
            isbn (str): The International Standard Book Number.
            available (bool): True if the book is available, False otherwise.
        """
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available = available
        print(f"Book '{self.title}' by {self.author} created.")

    def display_info(self):
        """
        Prints the details of the book.
        """
        status = "Available" if self.available else "Currently Borrowed"
        print(f"\n--- Book Details ---")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"ISBN: {self.isbn}")
        print(f"Status: {status}")
        print(f"--------------------")

    def borrow_book(self):
        """
        Changes the book's status to 'borrowed' if available.
        """
        if self.available:
            self.available = False
            print(f"'{self.title}' has been borrowed.")
```

```
        else:
            print(f"'{self.title}' is already borrowed.")

    def return_book(self):
        """
        Changes the book's status to 'available' if borrowed.
        """
        if not self.available:
            self.available = True
            print(f"'{self.title}' has been returned.")
        else:
            print(f"'{self.title}' is already available.")

if __name__ == "__main__":
    print("--- Demonstrating Classes and Methods ---")

    # Create instances of the Book class
    book1 = Book("Python Crash Course", "Eric Matthes", "978-1593279288")
    book2 = Book("Clean Code", "Robert C. Martin", "978-0132350884",
available=False)

    # Display initial information
    book1.display_info()
    book2.display_info()

    # Demonstrate methods
    book1.borrow_book()
    book1.display_info()

    book1.borrow_book() # Try to borrow again

    book2.return_book()
    book2.display_info()

    book2.return_book() # Try to return again
```

Input

No direct user input is required. The program demonstrates class and method usage with hardcoded values.

Expected Output

```
--- Demonstrating Classes and Methods ---
Book 'Python Crash Course' by Eric Matthes created.
Book 'Clean Code' by Robert C. Martin created.

--- Book Details ---
Title: Python Crash Course
Author: Eric Matthes
ISBN: 978-1593279288
Status: Available
--------------------

--- Book Details ---
Title: Clean Code
Author: Robert C. Martin
ISBN: 978-0132350884
Status: Currently Borrowed
--------------------
'Python Crash Course' has been borrowed.

--- Book Details ---
Title: Python Crash Course
```

```
Author: Eric Matthes
ISBN: 978-1593279288
Status: Currently Borrowed
--------------------
'Python Crash Course' is already borrowed.
'Clean Code' has been returned.

--- Book Details ---
Title: Clean Code
Author: Robert C. Martin
ISBN: 978-0132350884
Status: Available
--------------------
'Clean Code' is already available.
```

Laboratory 14: Python Program for Operator overloading

Aim

To write a Python program that demonstrates operator overloading, allowing custom behavior for standard operators (like `+`, `-`, `*`, etc.) when applied to instances of user-defined classes.

Procedure

1. **Define a Class:** Create a class (e.g., `Vector`, `Point`, `ComplexNumber`) that will have custom operator behavior.
2. **Implement Special Methods (Dunder Methods):** Override relevant special methods (also known as "dunder" methods, e.g., `__add__`, `__sub__`, `__mul__`) within the class.
    - `__add__(self, other)`: Implements the `+` operator.
    - `__sub__(self, other)`: Implements the `-` operator.
    - `__mul__(self, other)`: Implements the `*` operator (can be for scalar multiplication or dot product, depending on context).
    - `__str__(self)` or `__repr__(self)`: For a user-friendly string representation of the object.
3. **Create Objects:** Create instances of the class.
4. **Perform Operations:** Use the overloaded operators on the objects and observe the custom behavior.

Source Code

```
# lab14_operator_overloading.py

class Vector:
    """
    Represents a 2D vector and demonstrates operator overloading for basic
arithmetic.
    """
    def __init__(self, x, y):
        """
        Initializes a Vector object with x and y components.
        """
        self.x = x
        self.y = y

    def __str__(self):
        """
        Provides a user-friendly string representation of the Vector.
        """
        return f"Vector({self.x}, {self.y})"

    def __add__(self, other):
        """
        Overloads the '+' operator for vector addition.
        Adds corresponding components of two vectors.
        """
        if isinstance(other, Vector):
            return Vector(self.x + other.x, self.y + other.y)
        else:
            raise TypeError("Can only add a Vector to another Vector.")

    def __sub__(self, other):
        """
        Overloads the '-' operator for vector subtraction.
        Subtracts corresponding components of two vectors.
        """
```

```
        if isinstance(other, Vector):
            return Vector(self.x - other.x, self.y - other.y)
        else:
            raise TypeError("Can only subtract a Vector from another
Vector.")

    def __mul__(self, scalar):
        """
        Overloads the '*' operator for scalar multiplication.
        Multiplies each component of the vector by a scalar.
        """
        if isinstance(scalar, (int, float)):
            return Vector(self.x * scalar, self.y * scalar)
        else:
            raise TypeError("Can only multiply a Vector by a scalar (int or
float).")

    def __rmul__(self, scalar):
        """
        Overloads the '*' operator for right-hand side scalar multiplication
(e.g., 2 * vector).
        """
        return self.__mul__(scalar) # Delegate to the __mul__ method

    def __eq__(self, other):
        """
        Overloads the '==' operator for vector equality comparison.
        """
        if isinstance(other, Vector):
            return self.x == other.x and self.y == other.y
        return False

if __name__ == "__main__":
    print("--- Demonstrating Operator Overloading ---")

    v1 = Vector(2, 3)
    v2 = Vector(5, 1)
    v3 = Vector(2, 3)

    print(f"Vector 1: {v1}")
    print(f"Vector 2: {v2}")

    # Addition (+)
    v_sum = v1 + v2
    print(f"Vector 1 + Vector 2: {v_sum}")

    # Subtraction (-)
    v_diff = v1 - v2
    print(f"Vector 1 - Vector 2: {v_diff}")

    # Scalar Multiplication (*)
    v_scaled = v1 * 3
    print(f"Vector 1 * 3: {v_scaled}")

    # Right-hand side Scalar Multiplication (using __rmul__)
    v_scaled_r = 4 * v2
    print(f"4 * Vector 2: {v_scaled_r}")

    # Equality (==)
    print(f"Vector 1 == Vector 2: {v1 == v2}")
    print(f"Vector 1 == Vector 3: {v1 == v3}")

    # Error handling for invalid operations
    try:
        invalid_sum = v1 + "hello"
    except TypeError as e:
```

```
        print(f"\nError caught: {e}")

    try:
        invalid_mul = v1 * "scalar"
    except TypeError as e:
        print(f"Error caught: {e}")
```

Input

No direct user input is required. The program demonstrates operator overloading with hardcoded `Vector` objects and operations.

Expected Output

```
--- Demonstrating Operator Overloading ---
Vector 1: Vector(2, 3)
Vector 2: Vector(5, 1)
Vector 1 + Vector 2: Vector(7, 4)
Vector 1 - Vector 2: Vector(-3, 2)
Vector 1 * 3: Vector(6, 9)
4 * Vector 2: Vector(20, 4)
Vector 1 == Vector 2: False
Vector 1 == Vector 3: True

Error caught: Can only add a Vector to another Vector.
Error caught: Can only multiply a Vector by a scalar (int or float).
```

Laboratory 15: Program using polymorphism, abstract classes

Aim

To write a Python program that demonstrates polymorphism and the concept of abstract classes using the `abc` module. This will involve defining a common interface (abstract base class) and then implementing that interface in multiple concrete subclasses, showcasing how different objects can respond to the same method call in their own unique ways.

Procedure

1. **Import `ABC` and `abstractmethod`:** From the `abc` module.
2. **Define an Abstract Base Class (ABC):**
    - Inherit from `ABC`.
    - Decorate one or more methods with `@abstractmethod`. These methods must be implemented by concrete subclasses.
3. **Define Concrete Subclasses:**
    - Inherit from the ABC.
    - Implement all abstract methods defined in the parent ABC.
    - Add any specific attributes or methods relevant to the subclass.
4. **Demonstrate Polymorphism:**
    - Create instances of different concrete subclasses.
    - Store these instances in a collection (e.g., a list).
    - Iterate through the collection and call the common method (the one defined as abstract in the ABC). Observe how each object performs its specific implementation.

Source Code

```
# lab15_polymorphism_abstract_classes.py

from abc import ABC, abstractmethod

# 1. Define an Abstract Base Class (ABC)
class Shape(ABC):
    """
    An abstract base class representing a generic geometric shape.
    It defines common methods that all concrete shapes must implement.
    """
    def __init__(self, name):
        self.name = name

    @abstractmethod
    def area(self):
        """
        Abstract method to calculate the area of the shape.
        Must be implemented by subclasses.
        """
        pass # No implementation here

    @abstractmethod
    def perimeter(self):
        """
        Abstract method to calculate the perimeter of the shape.
        Must be implemented by subclasses.
        """
        pass # No implementation here

    def display_info(self):
```

```
        """
        A concrete method in the ABC that can be used by subclasses.
        """
        print(f"--- {self.name} ---")
        print(f"Area: {self.area():.2f}")
        print(f"Perimeter: {self.perimeter():.2f}")
        print("--------------------")

# 2. Define Concrete Subclasses
class Circle(Shape):
    """
    A concrete class representing a circle, inheriting from Shape.
    """
    def __init__(self, radius):
        super().__init__("Circle") # Call parent constructor
        self.radius = radius

    def area(self):
        """Calculates the area of the circle."""
        return 3.14159 * self.radius ** 2

    def perimeter(self):
        """Calculates the perimeter (circumference) of the circle."""
        return 2 * 3.14159 * self.radius

class Rectangle(Shape):
    """
    A concrete class representing a rectangle, inheriting from Shape.
    """
    def __init__(self, length, width):
        super().__init__("Rectangle") # Call parent constructor
        self.length = length
        self.width = width

    def area(self):
        """Calculates the area of the rectangle."""
        return self.length * self.width

    def perimeter(self):
        """Calculates the perimeter of the rectangle."""
        return 2 * (self.length + self.width)

class Triangle(Shape):
    """
    A concrete class representing a right-angled triangle, inheriting from
Shape.
    (For simplicity, assuming a right-angled triangle for perimeter
calculation)
    """
    def __init__(self, base, height):
        super().__init__("Triangle") # Call parent constructor
        self.base = base
        self.height = height
        # Calculate hypotenuse for perimeter (Pythagorean theorem)
        self.hypotenuse = (base**2 + height**2)**0.5

    def area(self):
        """Calculates the area of the triangle."""
        return 0.5 * self.base * self.height

    def perimeter(self):
        """Calculates the perimeter of the right-angled triangle."""
        return self.base + self.height + self.hypotenuse

if __name__ == "__main__":
    print("--- Demonstrating Polymorphism and Abstract Classes ---")
```

```
    # Create instances of different concrete subclasses
    circle1 = Circle(radius=5)
    rectangle1 = Rectangle(length=10, width=4)
    triangle1 = Triangle(base=3, height=4)

    # Store them in a list (demonstrating polymorphism)
    shapes = [circle1, rectangle1, triangle1]

    # Iterate through the collection and call common methods
    # Each object responds to 'area()' and 'perimeter()' in its own way
    print("\n--- Processing Shapes Polymorphically ---")
    for shape in shapes:
        shape.display_info() # Calling a concrete method from ABC
        # print(f"{shape.name} Area: {shape.area():.2f}, Perimeter:
{shape.perimeter():.2f}")

    # Attempting to instantiate the abstract class directly will raise an
error
    try:
        abstract_shape = Shape("Generic")
    except TypeError as e:
        print(f"\nError caught: {e}")
        print("Cannot instantiate an abstract class directly.")
```

Input

No direct user input is required. The program demonstrates polymorphism and abstract class usage with hardcoded `Shape` subclasses and their instances.

Expected Output

```
--- Demonstrating Polymorphism and Abstract Classes ---

--- Processing Shapes Polymorphically ---
--- Circle ---
Area: 78.54
Perimeter: 31.42
--------------------
--- Rectangle ---
Area: 40.00
Perimeter: 28.00
--------------------
--- Triangle ---
Area: 6.00
Perimeter: 12.00
--------------------

Error caught: Can't instantiate abstract class Shape with abstract methods
area, perimeter
Cannot instantiate an abstract class directly.
```