Python

LAB MANUAL

Dr. Muhammad Siddique

Lab Manual Programming Fundamentals (Using Python)

Dr. Muhammad Siddique

Course Overview

This lab manual provides a hands-on approach to learning Python programming for students in the Programming Fundamentals course. The labs are designed to introduce students to basic Python concepts, problem-solving strategies, and programming skills.

List of Labs for 16 Weeks

1. Week 1: Introduction to Python - Setting up the environment, basics of Python syntax
2. Week 2: Variables, Data Types, and Basic Input/Output
3. Week 3: Control Structures - Conditional Statements (if, else, elif)
4. Week 4: Loops - For and While Loops
5. Week 5: Functions - Definition and Calling
6. Week 6: Lists, Tuples, and Dictionaries
7. Week 7: Working with Strings
8. Week 8: File Handling
9. Week 9: Exception Handling
10. Week 10: Python Modules and Libraries
11. Week 11: Object-Oriented Programming (OOP) - Classes and Objects
12. Week 12: Inheritance and Polymorphism
13. Week 13: Introduction to NumPy for Data Handling
14. Week 14: Introduction to Pandas for Data Analysis
15. Week 15: Working on a Project (Part 1) - Problem Definition, Planning
16. Week 16: Working on a Project (Part 2) - Implementation, Testing

Week 1: Introduction to Python

Objective:

- Understand Python's environment setup, syntax, and basic operations.

Lab Task 1: Setting Up Python Environment

- Download and install Python from python.org.
- Install an Integrated Development Environment (IDE) like PyCharm or VS Code.

Lab Task 2: Running a Python Script

- Write a simple Python program that prints "Hello, World!" to the console.
- Use comments to describe the code.

Sample Code:

```
# This is a simple Python program to print a message
print("Hello, World!")
```

Lab Task 3: Exploring Python Syntax

- Experiment with indentation and block structure in Python.
- Try running some simple mathematical operations.

Sample Code:

```
# Simple addition and multiplication
a = 10
b = 5
print("Sum:", a + b)
print("Product:", a * b)
```

Week 2: Variables, Data Types, and Basic Input/Output

Objective:

- Understand variables, data types, and how to take input and print output in Python.

Lab Task 1: Declaring Variables and Using Different Data Types

- Write Python code to demonstrate the usage of integers, floats, and strings.
- Explain how Python handles dynamic typing.

Sample Code:

```python
# Integer, float, and string variables
x = 10              # Integer
y = 3.14            # Float
name = "Alice"      # String

print("Integer:", x)
print("Float:", y)
print("String:", name)
```

Lab Task 2: Taking Input and Displaying Output

- Write a program that takes user input and displays a message using the input.
- Use the `input()` function to take user input and `print()` function for output.

Sample Code:

```python
# Taking user input and displaying it
user_name = input("Enter your name: ")
print("Hello, " + user_name + "!")
```

Lab Task 3: Type Conversion

- Demonstrate how to convert data types in Python.
- Write code to convert strings to integers and floats.

Sample Code:

```
# Type conversion
num_str = "15"
num_int = int(num_str)    # Convert string to integer
num_float = float(num_str)  # Convert string to float

print("Integer:", num_int)
print("Float:", num_float)
```

Week 3: Control Structures - Conditional Statements (if, else, elif)

Objective:

- Understand how to use conditional statements in Python.
- Learn how to implement decision-making logic in Python programs using `if`, `else`, and `elif`.

Lab Task 1: Simple if Statement

- Write a Python program that checks if a number entered by the user is positive.
- Use an `if` statement to display a message if the condition is met.

Sample Code:

```
# Program to check if a number is positive
number = int(input("Enter a number: "))
if number > 0:
    print("The number is positive.")
```

Lab Task 2: if-else Statement

- Modify the previous program to also check if the number is negative or zero using an `else` statement.

Sample Code:

```
# Program to check if a number is positive or negative
number = int(input("Enter a number: "))
if number > 0:
    print("The number is positive.")
else:
    print("The number is negative or zero.")
```

Lab Task 3: if-elif-else Statement

- Extend the program to differentiate between positive, negative, and zero values using `elif`.

Sample Code:

```
# Program to check if a number is positive, negative, or zero
number = int(input("Enter a number: "))
if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

Lab Task 4: Nested if Statements

- Write a program that checks if a number is positive and even. If it's positive, use a nested `if` to check if it's also even.

Sample Code:

```
# Program to check if a number is positive and even
number = int(input("Enter a number: "))
if number > 0:
    print("The number is positive.")
    if number % 2 == 0:
        print("The number is even.")
    else:
        print("The number is odd.")
```

Lab Task 5: Using Logical Operators with if Statements

- Write a program that checks if a number is within a specific range (e.g., 10 to 50) using logical operators.
- Use the `and` operator to combine conditions.

Sample Code:

```python
# Program to check if a number is between 10 and 50
number = int(input("Enter a number: "))
if number >= 10 and number <= 50:
    print("The number is within the range 10 to 50.")
else:
    print("The number is outside the range.")
```

Week 4: Loops - For and While Loops

Objective:

- Understand how to use `for` and `while` loops in Python.
- Learn to implement repetition and iteration in Python programs.

Lab Task 1: Using a `for` Loop

- Write a Python program that prints numbers from 1 to 10 using a `for` loop.

Sample Code:

```
# Program to print numbers from 1 to 10 using a for loop
for i in range(1, 11):
    print(i)
```

Lab Task 2: Using a `while` Loop

- Write a Python program that prints numbers from 1 to 10 using a `while` loop.

Sample Code:

```
# Program to print numbers from 1 to 10 using a while loop
i = 1
while i <= 10:
    print(i)
    i += 1
```

Lab Task 3: Summing Numbers using a Loop

- Write a program that calculates the sum of the first 10 natural numbers using a `for` loop.

Sample Code:

```
# Program to sum the first 10 natural numbers
sum = 0
for i in range(1, 11):
    sum += i
print("The sum is:", sum)
```

Lab Task 4: Using a `while` Loop for User Input

- Write a Python program that repeatedly asks the user to enter a number until the number 0 is entered.

Sample Code:

```
# Program to repeatedly ask for input until 0 is entered
number = int(input("Enter a number (0 to stop): "))
while number != 0:
    print("You entered:", number)
    number = int(input("Enter a number (0 to stop): "))
```

Lab Task 5: Nested Loops

- Write a program to print a 5x5 multiplication table using nested `for` loops.

Sample Code:

```
# Program to print a 5x5 multiplication table
for i in range(1, 6):
    for j in range(1, 6):
        print(i * j, end="\t")
    print()  # Move to the next line
```

Week 5: Functions - Definition and Calling

Objective:

- Understand how to define and call functions in Python.
- Learn how to pass arguments to functions and return values.

Lab Task 1: Defining a Simple Function

- Write a Python function that takes no parameters and prints "Hello, Python!" when called.

  Sample Code:

```
# Simple function that prints a message
def greet():
    print("Hello, Python!")

# Calling the function
greet()
```

Lab Task 2: Function with Parameters

- Write a function that takes two numbers as parameters and prints their sum.

  Sample Code:

```
# Function that takes two numbers and prints their sum
def add_numbers(a, b):
    sum = a + b
    print("Sum:", sum)

# Calling the function
add_numbers(5, 10)
```

Lab Task 3: Function with Return Value

- Write a function that takes two numbers, multiplies them, and returns the result. Then call the function and print the returned value.

Sample Code:

```
# Function that returns the product of two numbers
def multiply_numbers(a, b):
    return a * b

# Calling the function and printing the result
result = multiply_numbers(4, 5)
print("Product:", result)
```

Lab Task 4: Function with Default Parameters

- Write a function that takes a name as a parameter and prints a greeting. If no name is provided, it should print a default greeting.

Sample Code:

```
# Function with a default parameter
def greet(name="Guest"):
    print("Hello,", name + "!")

# Calling the function with and without a parameter
greet("Alice")
greet()
```

Lab Task 5: Function with Keyword Arguments

- Write a function that calculates the area of a rectangle. Pass the values using keyword arguments.

Sample Code:

```
# Function that calculates the area of a rectangle using keyword arguments
def calculate_area(length=1, width=1):
    return length * width

# Calling the function with keyword arguments
area = calculate_area(length=5, width=3)
print("Area of rectangle:", area)
```

Week 6: Lists, Tuples, and Dictionaries

Objective:

- Understand how to use and manipulate lists, tuples, and dictionaries in Python.
- Learn the differences between mutable (lists, dictionaries) and immutable (tuples) data structures.

Lab Task 1: Working with Lists

- Write a program that creates a list of 5 integers, adds a new number to the list, and prints the updated list.

  Sample Code:

```
# List of integers
numbers = [10, 20, 30, 40, 50]

# Adding a number to the list
numbers.append(60)

# Printing the updated list
print("Updated list:", numbers)
```

Lab Task 2: List Operations

- Write a Python program that performs the following operations on a list:
  1. Insert a value at a specific index.
  2. Remove an element by value.
  3. Sort the list in ascending order.

  Sample Code:

```
# List operations
fruits = ["apple", "banana", "cherry"]

# Insert a value at index 1
fruits.insert(1, "orange")
```

```
# Remove an element by value
fruits.remove("banana")

# Sort the list
fruits.sort()

# Printing the updated list
print("Updated fruits list:", fruits)
```

Lab Task 3: Working with Tuples

- Write a Python program that creates a tuple with 5 elements, accesses an element by index, and prints the tuple in reverse order.

Sample Code:

```
# Tuple of elements
my_tuple = (1, 2, 3, 4, 5)

# Accessing an element by index
print("Element at index 2:", my_tuple[2])

# Printing the tuple in reverse order
print("Reversed tuple:", my_tuple[::-1])
```

Lab Task 4: Dictionary Operations

- Write a Python program that creates a dictionary, adds a new key-value pair, updates an existing value, and prints the dictionary.

Sample Code:

```
# Dictionary of student grades
grades = {"Alice": 85, "Bob": 90, "Charlie": 78}

# Adding a new key-value pair
grades["David"] = 88
```

```
# Updating an existing value
grades["Alice"] = 92

# Printing the dictionary
print("Updated grades:", grades)
```

Lab Task 5: Iterating through a Dictionary

- Write a program that iterates over the dictionary from the previous task and prints each key-value pair.

  Sample Code:

```
# Iterating through a dictionary
for student, grade in grades.items():
    print(f"{student}: {grade}")
```

Week 7: String Manipulation and Operations

Objective:

- Learn how to create, manipulate, and perform operations on strings in Python.
- Understand various string methods and functions for common string operations.

Lab Task 1: Basic String Operations

- Write a Python program that takes a string input from the user and performs the following operations:
    1. Print the string in uppercase.
    2. Print the string in lowercase.
    3. Calculate the length of the string.

Sample Code:

```
# Taking string input
user_string = input("Enter a string: ")

# Printing the string in uppercase
print("Uppercase:", user_string.upper())

# Printing the string in lowercase
print("Lowercase:", user_string.lower())

# Calculating the length of the string
print("Length of string:", len(user_string))
```

Lab Task 2: String Slicing

- Write a Python program that takes a string input and performs the following slicing operations:
    1. Print the first 5 characters of the string.
    2. Print the last 3 characters of the string.

3. Print every second character in the string.

Sample Code:

```
# String slicing operations
user_string = input("Enter a string: ")

# First 5 characters
print("First 5 characters:", user_string[:5])

# Last 3 characters
print("Last 3 characters:", user_string[-3:])

# Every second character
print("Every second character:", user_string[::2])
```

Lab Task 3: String Concatenation and Repetition

- Write a program that takes two strings as input and performs the following:
  1. Concatenate the two strings.
  2. Repeat the first string 3 times.

Sample Code:

```
# String concatenation and repetition
str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")

# Concatenating the strings
print("Concatenated string:", str1 + str2)

# Repeating the first string 3 times
print("Repeated string:", str1 * 3)
```

Lab Task 4: Checking Substrings

- Write a Python program that checks if a given substring exists within a string. If the substring is found, print its position.

Sample Code:

```
# Checking for a substring
main_string = input("Enter the main string: ")
sub_string = input("Enter the substring to find: ")

if sub_string in main_string:
    print(f"Substring found at index {main_string.find(sub_string)}")
else:
    print("Substring not found.")
```

Lab Task 5: String Formatting

- Write a Python program that takes a user's name and age as input and prints a formatted message like: "Hello, [name]! You are [age] years old."

Sample Code:

```
# String formatting
name = input("Enter your name: ")
age = int(input("Enter your age: "))

# Using formatted string
print(f"Hello, {name}! You are {age} years old.")
```

Week 8: File Handling in Python

Objective:

- Learn how to work with files in Python (reading, writing, appending).
- Understand how to handle exceptions while working with files.

Lab Task 1: Writing to a File

- Write a Python program that creates a text file named `output.txt` and writes the string "Hello, File Handling!" to the file.

Sample Code:

```
# Writing to a file
with open("output.txt", "w") as file:
    file.write("Hello, File Handling!")
```

Lab Task 2: Reading from a File

- Write a Python program that reads the contents of the file `output.txt` and prints it to the console.

Sample Code:

```
# Reading from a file
with open("output.txt", "r") as file:
    content = file.read()
    print(content)
```

Lab Task 3: Appending to a File

- Write a program that appends the string "This is an appended line." to the `output.txt` file.

Sample Code:

```
# Appending to a file
with open("output.txt", "a") as file:
    file.write("\nThis is an appended line.")
```

Lab Task 4: Reading File Line by Line

- Write a program that reads the `output.txt` file line by line and prints each line.

Sample Code:

```
# Reading a file line by line
with open("output.txt", "r") as file:
    for line in file:
        print(line.strip())
```

Lab Task 5: Handling File Not Found Exception

- Write a Python program that attempts to open a non-existing file `nonexistent.txt` and handles the `FileNotFoundError` exception gracefully.

Sample Code:

```
# Handling FileNotFoundError
try:
    with open("nonexistent.txt", "r") as file:
        content = file.read()
except FileNotFoundError:
    print("File not found. Please check the file name.")
```

Week 9: Exception Handling in Python

Objective:

- Learn how to handle errors and exceptions in Python programs.
- Understand the use of `try`, `except`, `finally`, and `else` clauses in exception handling.

Lab Task 1: Basic Exception Handling

- Write a Python program that divides two numbers entered by the user. Use a `try-except` block to handle the `ZeroDivisionError` exception.

  Sample Code:

```
# Handling ZeroDivisionError
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))
    result = numerator / denominator
    print(f"Result: {result}")
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
```

Lab Task 2: Handling Multiple Exceptions

- Write a program that prompts the user for two numbers and handles both `ZeroDivisionError` and `ValueError` exceptions.

  Sample Code:

```
# Handling multiple exceptions
try:
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))
    result = num1 / num2
    print(f"Result: {result}")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Please enter a valid integer.")
```

Lab Task 3: Using `else` with `try-except`

- Modify the program from Task 2 to include an `else` block that prints a success message if no exceptions are raised.

Sample Code:

```
# Using else with try-except
try:
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))
    result = num1 / num2
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Please enter a valid integer.")
else:
    print(f"Result: {result}. Operation was successful.")
```

Lab Task 4: Using `finally` Block

- Write a Python program that performs a file operation (open, write, and close a file) and includes a `finally` block to ensure that the file is closed, even if an error occurs.

Sample Code:

```
# Using finally to close a file
try:
    file = open("testfile.txt", "w")
    file.write("Writing some data into the file.")
except IOError:
    print("Error: File operation failed.")
finally:
    file.close()
    print("File closed successfully.")
```

Lab Task 5: Raising Custom Exceptions

- Write a Python program that raises a custom exception if a user enters a negative number. Handle this custom exception in the `except` block.

Sample Code:

```
# Custom exception for negative numbers
class NegativeNumberError(Exception):
    pass

try:
    num = int(input("Enter a positive number: "))
    if num < 0:
        raise NegativeNumberError("Negative number entered.")
except NegativeNumberError as e:
    print(e)
```

Week 10: Functions in Python

Objective:

- Understand how to define and call functions in Python.
- Learn the difference between positional, keyword, and default arguments.
- Learn how to use `return` to get values from functions.

Lab Task 1: Defining and Calling a Function

- Write a Python program that defines a function `greet` which takes a name as input and prints a greeting message.

Sample Code:

```
# Defining and calling a function
def greet(name):
    print(f"Hello, {name}! Welcome to Python functions.")

# Calling the function
greet("Alice")
```

Lab Task 2: Function with Return Value

- Write a Python program that defines a function `add` which takes two numbers as arguments, adds them, and returns the result.

Sample Code:

```
# Function with return value
def add(a, b):
    return a + b

# Calling the function
result = add(5, 7)
print(f"Sum: {result}")
```

Lab Task 3: Function with Default Argument

- Write a Python program that defines a function `introduce` which takes a name and an optional age (default value 25) and prints an introduction message.

Sample Code:

```
# Function with default argument
def introduce(name, age=25):
    print(f"My name is {name} and I am {age} years old.")

# Calling the function with and without the age argument
introduce("Bob", 30)
introduce("Alice")
```

Lab Task 4: Function with Keyword Arguments

- Write a Python program that defines a function `student_info` which takes a student's name and age as keyword arguments and prints their information.

Sample Code:

```
# Function with keyword arguments
def student_info(name, age):
    print(f"Student's Name: {name}, Age: {age}")

# Calling the function using keyword arguments
student_info(name="Charlie", age=22)
```

Lab Task 5: Function Returning Multiple Values

- Write a Python program that defines a function `calculate` which takes two numbers and returns both their sum and product.

Sample Code:

```
# Function returning multiple values
def calculate(x, y):
    return x + y, x * y
```

```python
# Calling the function
sum_result, product_result = calculate(4, 5)
print(f"Sum: {sum_result}, Product: {product_result}")
```

Week 11: Recursion in Python

Objective:

- Understand the concept of recursion and how to define recursive functions.
- Learn how to use base cases to avoid infinite recursion.
- Solve problems using recursive functions.

Lab Task 1: Factorial using Recursion

- Write a Python program that defines a recursive function to calculate the factorial of a given number.

Sample Code:

```
# Recursive function to calculate factorial
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n-1)

# Calling the function
num = 5
print(f"Factorial of {num} is {factorial(num)}")
```

Lab Task 2: Fibonacci Series using Recursion

- Write a Python program that defines a recursive function to return the n-th number in the Fibonacci series.

Sample Code:

```
# Recursive function for Fibonacci series
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-2) + fibonacci(n-1)

# Calling the function
n = 6
print(f"Fibonacci number at position {n} is {fibonacci(n)}")
```

Lab Task 3: Sum of Digits using Recursion

- Write a Python program that defines a recursive function to find the sum of digits of a given number.

Sample Code:

```
# Recursive function to find sum of digits
def sum_of_digits(n):
    if n == 0:
        return 0
    else:
        return n % 10 + sum_of_digits(n // 10)

# Calling the function
num = 1234
print(f"Sum of digits of {num} is {sum_of_digits(num)}")
```

Lab Task 4: Power Calculation using Recursion

- Write a Python program that defines a recursive function to calculate the power of a number (base raised to exponent).

Sample Code:

```
# Recursive function to calculate power
def power(base, exp):
    if exp == 0:
        return 1
    else:
        return base * power(base, exp-1)
```

```
# Calling the function
base = 2
exp = 3
print(f"{base} raised to the power of {exp} is {power(base, exp)}")
```

Lab Task 5: Recursive Function to Find GCD

- Write a Python program that defines a recursive function to find the greatest common divisor (GCD) of two numbers using the Euclidean algorithm.

Sample Code:

```
# Recursive function to find GCD using Euclidean algorithm
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

# Calling the function
a = 48
b = 18
print(f"GCD of {a} and {b} is {gcd(a, b)}")
```

Week 12: Object-Oriented Programming (OOP) in Python

Objective:

- Understand the basics of Object-Oriented Programming (OOP) concepts such as classes, objects, methods, and attributes.
- Learn how to define and use classes and objects in Python.
- Understand the concept of inheritance and method overriding.

Lab Task 1: Defining a Class and Creating Objects

- Write a Python program that defines a class `Student` with attributes `name` and `age`. Create an object of this class and print the student's information.

  Sample Code:

```
# Defining a class and creating an object
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Student Name: {self.name}, Age: {self.age}")

# Creating an object of Student class
student1 = Student("Alice", 20)
student1.display_info()
```

Lab Task 2: Class with Methods

- Write a Python program that defines a class `Circle` with an attribute `radius` and methods `get_area()` and `get_circumference()` to calculate and return the area and circumference of the circle.

  Sample Code:

```
# Class with methods for area and circumference
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def get_area(self):
        return 3.14159 * self.radius ** 2

    def get_circumference(self):
        return 2 * 3.14159 * self.radius

# Creating an object of Circle class
circle1 = Circle(5)
print(f"Area: {circle1.get_area()}")
print(f"Circumference: {circle1.get_circumference()}")
```

Lab Task 3: Inheritance in Python

- Write a Python program that defines a class `Person` with attributes `name` and `age`. Create a subclass `Employee` that inherits from `Person` and adds an attribute `salary`. Create an object of `Employee` and print its information.

  Sample Code:

```
# Inheritance in Python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Employee(Person):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

    def display_info(self):
        print(f"Employee Name: {self.name}, Age: {self.age}, Salary: {self

# Creating an object of Employee class
```

```
employee1 = Employee("Bob", 30, 50000)
employee1.display_info()
```

Lab Task 4: Method Overriding

- Write a Python program that defines a class `Animal` with a method `sound()`. Create a subclass `Dog` that overrides the `sound()` method to print "Bark". Create an object of `Dog` and call the `sound()` method.

  Sample Code:

```
# Method overriding
class Animal:
    def sound(self):
        print("Some generic animal sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

# Creating an object of Dog class
dog1 = Dog()
dog1.sound()
```

Lab Task 5: Encapsulation and Private Attributes

- Write a Python program that defines a class `BankAccount` with private attribute `balance`. Implement methods to deposit and withdraw money while ensuring the balance remains private.

  Sample Code:

```
# Encapsulation and private attributes
class BankAccount:
    def __init__(self):
        self.__balance = 0  # Private attribute

    def deposit(self, amount):
        self.__balance += amount
    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient balance")

    def get_balance(self):
        return self.__balance

# Creating an object of BankAccount class
account = BankAccount()
account.deposit(1000)
account.withdraw(500)
print(f"Current balance: {account.get_balance()}")
```

Week 13: File Handling in Python

Objective:

- Learn how to work with files in Python (read, write, append).
- Understand file modes (read, write, append).
- Handle errors during file operations.

Lab Task 1: Reading from a File

- Write a Python program to read the contents of a file called `data.txt` and display the contents line by line.

Sample Code:

```
# Reading from a file
try:
    with open('data.txt', 'r') as file:
        for line in file:
            print(line.strip())
except FileNotFoundError:
    print("File not found.")
```

Lab Task 2: Writing to a File

- Write a Python program to write user input to a file called `output.txt`. Take a string input from the user and save it to the file.

Sample Code:

```
# Writing to a file
with open('output.txt', 'w') as file:
    user_input = input("Enter text to write to file: ")
    file.write(user_input)
print("Data written to file.")
```

Lab Task 3: Appending to a File

- Write a Python program to append new data to an existing file `log.txt`. If the file does not exist, create it.

Sample Code:

```python
# Appending to a file
with open('log.txt', 'a') as file:
    log_entry = input("Enter log entry: ")
    file.write(log_entry + '\n')
print("Log entry added.")
```

Lab Task 4: Counting Words in a File

- Write a Python program to count the number of words in a file called `article.txt`.

Sample Code:

```python
# Counting words in a file
try:
    with open('article.txt', 'r') as file:
        text = file.read()
        words = text.split()
        print(f"Total words: {len(words)}")
except FileNotFoundError:
    print("File not found.")
```

Lab Task 5: Copying Content from One File to Another

- Write a Python program to copy the contents of `source.txt` into a new file `destination.txt`.

Sample Code:

```python
# Copying content from one file to another
try:
    with open('source.txt', 'r') as source_file:
        content = source_file.read()
    with open('destination.txt', 'w') as dest_file:
        dest_file.write(content)
    print("Content copied successfully.")
except FileNotFoundError:
    print("Source file not found.")
```

Week 14: Exception Handling in Python

Objective:

- Understand the concept of exceptions and errors in Python.
- Learn how to handle exceptions using `try`, `except`, `finally`, and `else`.
- Raise custom exceptions.
- Practice writing robust programs that handle errors gracefully.

Lab Task 1: Basic Exception Handling

- Write a Python program that takes two numbers as input and performs division. Use `try` and `except` to handle any division by zero errors.

  Sample Code:

```python
# Basic exception handling for division
try:
    num1 = int(input("Enter the numerator: "))
    num2 = int(input("Enter the denominator: "))
    result = num1 / num2
    print(f"Result: {result}")
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Invalid input. Please enter integers.")
```

Lab Task 2: Multiple Exceptions Handling

- Write a Python program that reads a number from a file called `number.txt`. Use exception handling to manage file not found and value errors.

  Sample Code:

```python
# Multiple exceptions handling
try:
    with open('number.txt', 'r') as file:
        num = int(file.read())
        print(f"Number from file: {num}")
except FileNotFoundError:
    print("Error: The file does not exist.")
except ValueError:
    print("Error: Could not convert data to an integer.")
```

Lab Task 3: Using `else` and `finally` in Exception Handling

- Write a Python program that opens a file, reads its contents, and prints them. Use `else` to print a success message and `finally` to close the file regardless of success or failure.

Sample Code:

```
# Using else and finally
try:
    file = open('data.txt', 'r')
    content = file.read()
except FileNotFoundError:
    print("Error: File not found.")
else:
    print("File read successfully.")
    print(content)
finally:
    file.close()
    print("File is closed.")
```

Lab Task 4: Raising Custom Exceptions

- Write a Python program that defines a function to check the validity of a user's age. If the age is less than 18, raise a custom exception called `UnderageError`.

Sample Code:

```
# Raising custom exceptions
class UnderageError(Exception):
    pass

def check_age(age):
    if age < 18:
        raise UnderageError("User is underage.")
    else:
        print("User is eligible.")

try:
    user_age = int(input("Enter your age: "))
    check_age(user_age)
except UnderageError as e:
    print(e)
except ValueError:
    print("Error: Please enter a valid number.")
```

Lab Task 5: Handling Exceptions in File Operations

- Write a Python program that attempts to read a file and print its contents. Use a `try-except-finally` block to handle possible file handling exceptions like file not found, permission errors, etc.

Sample Code:

```
# Handling exceptions in file operations
try:
    file = open('restricted_file.txt', 'r')
    content = file.read()
    print(content)
except FileNotFoundError:
    print("Error: File not found.")
except PermissionError:
    print("Error: Permission denied.")
finally:
    try:
        file.close()
    except NameError:
        print("File was never opened.")
```

Week 15: Working with Libraries and Modules in Python

Objective:

- Understand how to import and use built-in and external libraries in Python.
- Learn how to create and use custom modules.
- Explore common Python libraries such as `math`, `datetime`, and `random`.
- Install and use external libraries using `pip`.

Lab Task 1: Using the `math` Library

- Write a Python program that uses the `math` library to calculate the square root, power, and factorial of a number.

Sample Code:

```
import math

# Using math library functions
num = 5
sqrt_val = math.sqrt(num)
power_val = math.pow(num, 3)
factorial_val = math.factorial(num)

print(f"Square root of {num}: {sqrt_val}")
print(f"{num} raised to the power 3: {power_val}")
print(f"Factorial of {num}: {factorial_val}")
```

Lab Task 2: Working with `datetime` Module

- Write a Python program that uses the `datetime` module to get the current date and time, and formats it to display only the year, month, and day.

Sample Code:

```
from datetime import datetime

# Get the current date and time
current_datetime = datetime.now()

# Format and display the date
formatted_date = current_datetime.strftime("%Y-%m-%d")
print(f"Current date: {formatted_date}")
```

Lab Task 3: Generating Random Numbers using `random` Module

- Write a Python program that uses the `random` module to generate a list of 5 random integers between 1 and 100, and then selects a random element from the list.

Sample Code:

```
import random

# Generating random numbers
random_numbers = random.sample(range(1, 101), 5)
random_choice = random.choice(random_numbers)

print(f"Random numbers: {random_numbers}")
print(f"Randomly selected number: {random_choice}")
```

Lab Task 4: Creating and Using a Custom Module

- Create a custom module `mymodule.py` that contains a function `greet(name)`. Write a main Python program that imports the module and calls the `greet()` function.

Sample Code for `mymodule.py`:

```
# mymodule.py
def greet(name):
    print(f"Hello, {name}!")
```

Sample Code for Main Program:

```
# Main program
import mymodule

# Call the greet function from custom module
mymodule.greet("Alice")
```

Lab Task 5: Installing and Using External Libraries with `pip`

- Write a Python program that uses the `requests` library to fetch data from a URL and prints the status code of the response. (Ensure that the `requests` library is installed using `pip`.)

  Sample Code:

```
import requests

# Fetch data from a URL
response = requests.get('https://www.example.com')

# Print the status code of the response
print(f"Status code: {response.status_code}")
```

Week 15: Working with Libraries and Modules in Python

Objective:

- Understand how to import and use built-in and external libraries in Python.
- Learn how to create and use custom modules.
- Explore common Python libraries such as `math`, `datetime`, and `random`.
- Install and use external libraries using `pip`.

Lab Task 1: Using the `math` Library

- Write a Python program that uses the `math` library to calculate the square root, power, and factorial of a number.

Sample Code:

```
import math

# Using math library functions
num = 5
sqrt_val = math.sqrt(num)
power_val = math.pow(num, 3)
factorial_val = math.factorial(num)

print(f"Square root of {num}: {sqrt_val}")
print(f"{num} raised to the power 3: {power_val}")
print(f"Factorial of {num}: {factorial_val}")
```

Lab Task 2: Working with `datetime` Module

- Write a Python program that uses the `datetime` module to get the current date and time, and formats it to display only the year, month, and day.

Sample Code:

```
from datetime import datetime

# Get the current date and time
current_datetime = datetime.now()

# Format and display the date
formatted_date = current_datetime.strftime("%Y-%m-%d")
print(f"Current date: {formatted_date}")
```

Lab Task 3: Generating Random Numbers using `random` Module

- Write a Python program that uses the `random` module to generate a list of 5 random integers between 1 and 100, and then selects a random element from the list.

Sample Code:

```
import random

# Generating random numbers
random_numbers = random.sample(range(1, 101), 5)
random_choice = random.choice(random_numbers)

print(f"Random numbers: {random_numbers}")
print(f"Randomly selected number: {random_choice}")
```

Lab Task 4: Creating and Using a Custom Module

- Create a custom module `mymodule.py` that contains a function `greet(name)`. Write a main Python program that imports the module and calls the `greet()` function.

Sample Code for `mymodule.py`:

```
# mymodule.py
def greet(name):
    print(f"Hello, {name}!")
```

Sample Code for Main Program:

```
# Main program
import mymodule

# Call the greet function from custom module
mymodule.greet("Alice")
```

Lab Task 5: Installing and Using External Libraries with `pip`

- Write a Python program that uses the `requests` library to fetch data from a URL and prints the status code of the response. (Ensure that the `requests` library is installed using `pip`.)

Sample Code:

```
import requests

# Fetch data from a URL
response = requests.get('https://www.example.com')

# Print the status code of the response
print(f"Status code: {response.status_code}")
```

Week 16: Project – Bringing It All Together

Objective:

- Apply all the concepts learned throughout the course to build a complete Python project.
- Practice working with functions, file handling, exception handling, modules, and libraries.
- Strengthen problem-solving and logical thinking skills through a real-world application.

Lab Task 1: Problem Statement

- You are required to develop a mini project in Python that simulates a **Student Management System**. The system should allow a user to add, view, update, and delete student records. Each student record should contain the following fields: `Student ID`, `Name`, `Age`, `Course`.

Lab Task 2: Add New Student Record

- Write a function `add_student()` that prompts the user for student details (ID, Name, Age, and Course) and saves them to a file `students.txt`. If the file doesn't exist, create it.

  Sample Code:

```python
def add_student():
    with open('students.txt', 'a') as file:
        student_id = input("Enter Student ID: ")
        name = input("Enter Student Name: ")
        age = input("Enter Student Age: ")
        course = input("Enter Course: ")
        file.write(f"{student_id},{name},{age},{course}\n")
    print("Student record added.")
```

Lab Task 3: View All Student Records

- Write a function `view_students()` that reads and displays all student records from `students.txt`. Handle the case where the file might not exist.

Sample Code:

```
def view_students():
    try:
        with open('students.txt', 'r') as file:
            for line in file:
                student = line.strip().split(',')
                print(f"ID: {student[0]}, Name: {student[1]}, Age: {studen
    except FileNotFoundError:
        print("No student records found.")
```

Lab Task 4: Update a Student Record

- Write a function `update_student()` that allows the user to update a student's information based on their `Student ID`. Modify the corresponding record in `students.txt`.

Sample Code:

```
def update_student():
    student_id = input("Enter the Student ID to update: ")
    updated_records = []
    found = False
    try:
        with open('students.txt', 'r') as file:
            for line in file:
                student = line.strip().split(',')
                if student[0] == student_id:
                    name = input("Enter new Name: ")
                    age = input("Enter new Age: ")
                    course = input("Enter new Course: ")
                    updated_records.append(f"{student_id},{name},{age},{co
                    found = True
                else:
                    updated_records.append(line)
        with open('students.txt', 'w') as file:
            file.writelines(updated_records)

        if found:
            print("Student record updated.")
        else:
            print("Student ID not found.")
    except FileNotFoundError:
        print("No student records found.")
```

Lab Task 5: Delete a Student Record

- Write a function `delete_student()` that allows the user to delete a student's record based on their `Student ID`.

Sample Code:

```
def delete_student():
    student_id = input("Enter the Student ID to delete: ")
    updated_records = []
    found = False
    try:
        with open('students.txt', 'r') as file:
            for line in file:
                student = line.strip().split(',')
                if student[0] != student_id:
                    updated_records.append(line)
                else:
                    found = True

        with open('students.txt', 'w') as file:
            file.writelines(updated_records)

        if found:
            print("Student record deleted.")
        else:
            print("Student ID not found.")
    except FileNotFoundError:
```

```
print("No student records found.")
```

Week 16: Project – Bringing It All Together

Objective:

- Apply all the concepts learned throughout the course to build a complete Python project.
- Practice working with functions, file handling, exception handling, modules, and libraries.
- Strengthen problem-solving and logical thinking skills through a real-world application.

Lab Task 1: Problem Statement

- You are required to develop a mini project in Python that simulates a **Student Management System**. The system should allow a user to add, view, update, and delete student records. Each student record should contain the following fields: `Student ID`, `Name`, `Age`, `Course`.

Lab Task 2: Add New Student Record

- Write a function `add_student()` that prompts the user for student details (ID, Name, Age, and Course) and saves them to a file `students.txt`. If the file doesn't exist, create it.

Sample Code:

```
def add_student():
    with open('students.txt', 'a') as file:
        student_id = input("Enter Student ID: ")
        name = input("Enter Student Name: ")
        age = input("Enter Student Age: ")
        course = input("Enter Course: ")
        file.write(f"{student_id},{name},{age},{course}\n")
    print("Student record added.")
```

Lab Task 3: View All Student Records

- Write a function `view_students()` that reads and displays all student records from `students.txt`. Handle the case where the file might not exist.

Sample Code:

```
def view_students():
    try:
        with open('students.txt', 'r') as file:
            for line in file:
                student = line.strip().split(',')
                print(f"ID: {student[0]}, Name: {student[1]}, Age: {studen
    except FileNotFoundError:
        print("No student records found.")
```

Lab Task 4: Update a Student Record

- Write a function `update_student()` that allows the user to update a student's information based on their `Student ID`. Modify the corresponding record in `students.txt`.

Sample Code:

```
def update_student():
    student_id = input("Enter the Student ID to update: ")
    updated_records = []
    found = False
    try:
        with open('students.txt', 'r') as file:
            for line in file:
                student = line.strip().split(',')
                if student[0] == student_id:
                    name = input("Enter new Name: ")
                    age = input("Enter new Age: ")
                    course = input("Enter new Course: ")
                    updated_records.append(f"{student_id},{name},{age},{co
                    found = True
                else:
                    updated_records.append(line)
        with open('students.txt', 'w') as file:
            file.writelines(updated_records)

        if found:
            print("Student record updated.")
        else:
            print("Student ID not found.")
    except FileNotFoundError:
        print("No student records found.")
```

Lab Task 5: Delete a Student Record

- Write a function `delete_student()` that allows the user to delete a student's record based on their `Student ID`.

Sample Code:

```
def delete_student():
    student_id = input("Enter the Student ID to delete: ")
    updated_records = []
    found = False
    try:
        with open('students.txt', 'r') as file:
            for line in file:
                student = line.strip().split(',')
                if student[0] != student_id:
                    updated_records.append(line)
                else:
                    found = True

        with open('students.txt', 'w') as file:
            file.writelines(updated_records)

        if found:
            print("Student record deleted.")
        else:
            print("Student ID not found.")
    except FileNotFoundError:
```

```
print("No student records found.")
```