Week 2

Section 1: Variables and Comparison Operators

Week 2 introduces fundamental programming concepts including variables, operators, control structures, and data manipulation. The lab demonstrates practical applications of these concepts through various exercises.

Exercise 1: Basic Comparison Operations

The lab begins with variable assignment and comparison operators to understand how Python evaluates conditions.

```
# Variables Definition
x = 24  # First number for comparison
y = 51  # Second number for comparison

# Comparison Operators
a1 = x == y   # Equality: 24 == 51 → False
a2 = x != y   # Inequality: 24 != 51 → True
a3 = x > y    # Greater than: 24 > 51 → False
a4 = x < y    # Less than: 24 < 51 → True
a5 = x >= y   # Greater than or equal: 24 >= 51 → False
a6 = x <= y   # Less than or equal: 24 <= 51 → True
```

Output when run:

```
Comparison Operators
Equality:  False
Inequality:  True
Greater than:  False
Less than:  True
Greater than or equal to:  False
Less than or equal to:  True
```

Exercise 2: Demonstrating False Conditions

This exercise shows how changing variable values affects comparison results, demonstrating the dynamic nature of variables.

```
# Making equality operator return False
x = 10  # Change x to a value different from y (51)
a1 = x == y  # 10 == 51 → False

# Making inequality operator return False
```

```
x = 51  # Change x to equal y
a2 = x != y  # 51 != 51 → False
```

Output when run:

```
Equality: where x =  10 and y =  51  the condition is False
Inequality: where x =  51 and y =  51  the condition is False
```

Section 2: Logical Operators

Logical operators combine multiple boolean expressions, essential for complex decision-making in programs.

```
# Current values: x = 58, y = 51
b1 = x > 10 and y < 100  # AND: both conditions must be True
b2 = x > 10 or y < 100   # OR: at least one condition must be True
b3 = not(x > 10 and y < 100)  # NOT: inverts the result
```

Output when run:

```
Logical Operators
And:  True
Or:  True
Not:  False
```

Section 3: Conditional Statements

Exercise 1: Simple If Statements

```
# Test 1: Age qualifies as adult
age = 19
age_group = "child"  # Default value
if age > 18:  # Condition: 19 > 18 → True
    age_group = "adult"
    print(f"The age group is {age_group}")
```

Output when run:

```
The age group is adult
```

Exercise 2: If-Else Statements

```
# Wind speed example
wind_speed = 30
if wind_speed < 10:
    print("It is a calm day")
else:
    print("It is a windy day")
```

Output when run:

```
It is a windy day
```

Exercise 3: If-Elif-Else Statements

```
grade = 55
if grade < 50:
    print("You failed")
elif grade < 60:
    print("You passed")
elif grade < 70:
    print("You got a good pass")
else:
    print("You got an excellent pass")
```

Output when run:

```
You passed
```

Section 4: Lists and List Operations

Lists are fundamental data structures in Python for storing ordered collections of items.

Exercise 1: Basic List Operations

```
# Creating and accessing lists
city_list = ["Glasgow", "London", "Edinburgh"]
print(city_list)  # Output: ['Glasgow', 'London', 'Edinburgh']

# Accessing elements
print(city_list[2])   # Third element: "Edinburgh"
print(city_list[-1])  # Last element: "Edinburgh"

# List slicing
```

```
print(city_list[1:3])  # Elements from index 1 to 2: ['London',
'Edinburgh']
```

Exercise 2: List Modification

```
# Appending - adds element to the end
city_list.append("Manchester")
print(city_list)  # Output: ['Glasgow', 'London', 'Edinburgh',
'Manchester']

# Removing - removes first occurrence
city_list.remove("Manchester")
print(city_list)  # Output: ['Glasgow', 'London', 'Edinburgh']

# Inserting - adds element at specified index
city_list.insert(1, "Manchester")
print(city_list)  # Output: ['Glasgow', 'Manchester', 'London',
'Edinburgh']

# Replacing - changes element at specified index
city_list[1] = "Birmingham"
print(city_list)  # Output: ['Glasgow', 'Birmingham', 'London',
'Edinburgh']
```

Exercise 3: List Analysis

```
colours = ["beige", "indigo", "mangenta"]
print("The second item on the list is:", colours[1])  # Output: indigo

colours[1] = "pink"
print("The length of the colours list is: ", len(colours))  # Output: 3

# Checking for specific values
if colours[0] == "pink" or colours[1] == "pink" or colours[2] == "pink":
    print("Red is in the list")  # This executes since pink is present
```

Section 5: Loops

Exercise 1: For Loops with Lists

```
# Iterating through list elements
for city in city_list:
    print(city)
```

Output when run:

```
Glasgow
Birmingham
London
Edinburgh
```

Exercise 2: For Loops with Range and Break

```python
# Loop with break statement
for i in range(5):
    if i == 2:
        break
    print(i)
```

Output when run:

```
0
1
```

Exercise 3: Conditional Processing in Loops

```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for number in numbers:
    if number % 2 == 0:  # Check if even
        print(number)
```

Output when run:

```
2
4
6
8
10
```

Exercise 4: Accumulator Pattern

```python
sum_of_squares = 0
for n in range(1, 6):
    sum_of_squares += n ** 2
    print("The sum of squares is: ", sum_of_squares)
```

Output when run:

```
The sum of squares is:  1
The sum of squares is:  5
The sum of squares is:  14
The sum of squares is:  30
The sum of squares is:  55
```

Exercise 5: While Loops

```
countdown = 10
while countdown > 0:
    print(countdown)
    countdown -= 1
print("Liftoff!")
```

Output when run:

```
10
9
8
7
6
5
4
3
2
1
Liftoff!
```

Section 6: User Input and Data Conversion

Exercise 1: Age Classification Program

```
user_age = int(input("Please enter your age: "))
if user_age <= 17:
    print("You are a minor")
elif user_age >= 18 and user_age < 66:
    print("You are an adult")
else:
    print("You are a senior citizen")
```

Sample Input/Output:

```
Please enter your age: 45
You are an adult
```

Exercise 2: Temperature Conversion Program

A comprehensive program demonstrating multiple concepts working together:

```
temperature_unit = input("Please enter the temperature unit (C, F, K): ")
temperature = float(input("Please enter the temperature: "))
temperature_converter = input("Please enter the temperature unit you want
to convert to (C, F, K): ")

# Temperature conversion formulas
celcius_fahrenheit = (temperature * 9/5) + 32
celcius_kelvin = temperature + 273.15
fahrenheit_celcius = (temperature - 32) * 5/9
fahrenheit_kelvin = (temperature - 32) * 5/9 + 273.15
kelvin_celcius = temperature - 273.15
kelvin_fahrenheit = (temperature - 273.15) * 9/5 + 32

if temperature_unit == "C" and temperature_converter == "F":
    print(f"{temperature}C is {celcius_fahrenheit}F")
elif temperature_unit == "C" and temperature_converter == "K":
    print(f"{temperature}C is {celcius_kelvin}K")
# ... additional conversion cases
```

Sample Input/Output:

```
Please enter the temperature unit (C, F, K): C
Please enter the temperature: 25
Please enter the temperature unit you want to convert to (C, F, K): F
25.0C is 77.0F
```

Key Programming Concepts Demonstrated in Week 2

1. **Variables and Data Types**: Integer, float, string, boolean
2. **Comparison Operators**: ==, !=, >, <, >=, <=
3. **Logical Operators**: and, or, not
4. **Control Flow**: if, if-else, if-elif-else statements
5. **Data Structures**: Lists and list operations (append, remove, insert, slicing)
6. **Iteration**: for loops, while loops, break statements
7. **User Interaction**: input() function and type conversion
8. **String Formatting**: f-strings for output formatting
9. **Mathematical Operations**: Basic arithmetic and compound calculations

---

Week 3

Section 1: Introduction to Functions

Week 3 focuses on function definition, parameters, return values, and error handling. Functions are essential for code organization, reusability, and modularity.

Exercise 1: Basic Function with Parameters

```
def greet_friends(friend_list):
    """
    Greets each friend in the provided list.

    Parameters:
    friend_list (list): A list of friend names to greet
    """
    for name in friend_list:
        print(f"Hello {name}!")

# Test the function
friend_list = ["John", "Jane", "Jack"]
greet_friends(friend_list)
```

Output when run:

```
Hello John!
Hello Jane!
Hello Jack!
```

Exercise 2: Functions with Return Values

```
def calculate_tax(income, tax_rate):
    """
    Calculates tax amount based on income and tax rate.

    Parameters:
    income (float): The income amount
    tax_rate (float): The tax rate as a decimal

    Returns:
    float: The calculated tax amount
    """
    return income * tax_rate

print("Tax on £50,000 at 20% rate:", calculate_tax(50000, 0.2))
print("Tax on £30,000 at 15% rate:", calculate_tax(30000, 0.15))
```

Output when run:

```
Tax on £50,000 at 20% rate: 10000.0
Tax on £30,000 at 15% rate: 4500.0
```

Section 2: Complex Functions with Validation

Exercise 1: Compound Interest Calculator with Input Validation

```
def compound_interest(principal, duration, interest_rate):
    """
    Calculates compound interest with input validation.
    Formula: A = P(1 + r)^t
    """
    # Input validation
    if interest_rate < 0 or interest_rate > 1:
        print("Please enter a decimal number between 0 and 1")
        return None
    if duration < 0:
        print("Please enter a positive number of years")
        return None

    # Calculate compound interest for each year
    for year in range(1, duration + 1):
        total = principal * (1 + interest_rate) ** year
        print(f"The total amount of money earned by the investment in year
{year} is {total:.0f} £")

    return int(total)

# Test the function
final_amount = compound_interest(1000, 5, 0.03)
print("Final Amount:", final_amount)
```

Output when run:

```
The total amount of money earned by the investment in year 1 is 1030 £
The total amount of money earned by the investment in year 2 is 1061 £
The total amount of money earned by the investment in year 3 is 1093 £
The total amount of money earned by the investment in year 4 is 1126 £
The total amount of money earned by the investment in year 5 is 1159 £
Final Amount: 1159
```

Section 3: Testing with Assertions

Exercise 1: Function Testing

```
# Use assert to test function returns expected result
assert compound_interest(1000, 5, 0.03) == 1159
```

This assertion tests that our compound interest function returns the correct value. If the assertion fails, the program will stop with an AssertionError, indicating a bug in our function.

Section 4: Error Handling and Common Python Errors

Exercise 1: Common Error Types and Fixes

```
# Fixing Syntax Error - Missing quotes
print("Hello, World!")  # Correct: String properly quoted

# Fixing Name Error - Undefined variable
favorite_color = "Blue"  # Define variable first
print("My favorite color is", favorite_color)

# Fixing Value Error - Type conversion
number1 = "5"  # String
number2 = 3    # Integer
result = int(number1) + number2  # Convert string to int
print("The sum is:", result)

# Fixing Index Error - Invalid list index
fruits = ["apple", "banana", "cherry"]
print(fruits[1])  # Access valid index

# Fixing Indentation Error - Proper code block indentation
time = 11
if time < 12:
    print("Good morning!")  # Properly indented
```

Output when run:

```
Hello, World!
My favorite color is Blue
The sum is: 8
banana
Good morning!
```

Section 5: Interactive Console Application

Exercise 1: To-Do List Manager

A complete interactive program demonstrating advanced concepts:

```
# Global data structure
tasks = []

def add_task():
    """Add a task to the list"""
    task = input("Enter the task you want to add: ")
    tasks.append(task)
    print(f"Task '{task}' added.")

def view_tasks():
    """Display all current tasks"""
    if not tasks:
        print("No tasks in the list.")
    else:
        print("Current tasks:")
        for i, task in enumerate(tasks, start=1):
            print(f"{i}. {task}")

def remove_task():
    """Remove a task by number with error handling"""
    view_tasks()
    if not tasks:
        return
    try:
        task_number = int(input("Enter the task number to remove: "))
        if 1 <= task_number <= len(tasks):
            removed = tasks.pop(task_number - 1)
            print(f"Task '{removed}' removed.")
        else:
            print("Invalid task number.")
    except ValueError:
        print("Please enter a valid number.")

# Main program loop
while True:
    print("\nTo-Do List Manager")
    print("1. Add a task")
    print("2. View tasks")
    print("3. Remove a task")
    print("4. Quit")

    choice = input("Enter your choice: ")

    if choice == "1":
        add_task()
    elif choice == "2":
        view_tasks()
    elif choice == "3":
        remove_task()
    elif choice == "4":
        print("Exiting program. Goodbye!")
```

```
            break
        else:
            print("Invalid choice. Please try again.")
```

Sample Program Interaction:

```
To-Do List Manager
1. Add a task
2. View tasks
3. Remove a task
4. Quit
Enter your choice: 1
Enter the task you want to add: Buy groceries
Task 'Buy groceries' added.

To-Do List Manager
1. Add a task
2. View tasks
3. Remove a task
4. Quit
Enter your choice: 2
Current tasks:
1. Buy groceries
```

Key Programming Concepts Demonstrated in Week 3

1. **Function Definition**: Creating reusable code blocks with def keyword
2. **Function Parameters**: Passing data into functions for processing
3. **Return Values**: Functions returning calculated results
4. **Input Validation**: Checking user input for validity before processing
5. **Error Handling**: Using try-except blocks to handle runtime errors
6. **Assertions**: Testing function behavior with assert statements
7. **Global Variables**: Sharing data across functions
8. **Interactive Programs**: Creating menu-driven console applications
9. **List Manipulation**: Advanced list operations with user interaction
10. **Program Flow Control**: While loops for continuous program execution
11. **String Formatting**: Advanced f-string usage for user feedback
12. **Mathematical Calculations**: Implementing financial formulas in code

Advanced Concepts Introduced

Function Documentation

- **Docstrings**: Multi-line strings describing function purpose and parameters
- **Type Hints**: Indicating expected parameter and return types in comments
- **Parameter Documentation**: Describing what each parameter represents

Error Types and Handling

- **SyntaxError**: Incorrect Python syntax
- **NameError**: Using undefined variables
- **ValueError**: Invalid value for operation (e.g., converting non-numeric string to int)
- **IndexError**: Accessing invalid list indices
- **IndentationError**: Incorrect code block indentation

Program Architecture

- **Modular Design**: Breaking programs into focused functions
- **Separation of Concerns**: Each function has a single responsibility
- **User Interface Design**: Creating intuitive menu systems
- **Data Persistence**: Maintaining program state throughout execution

Best Practices Demonstrated

- **Input Validation**: Always check user input before processing
- **Error Messages**: Provide clear, helpful error messages
- **Function Documentation**: Document function purpose and usage
- **Code Organization**: Group related functionality together
- **User Feedback**: Confirm actions and provide status updates

---