Python Lecture 2: Practice Assignments on Arithmetic, Relational, Logical Operators, and Branching

Instructions

Complete the following assignments to practice the concepts covered in Lecture 2, including arithmetic operators, operator precedence, relational operators, logical operators, and branching structures (if, if-else, if-elif-else). Write, run, and test your code in a Python environment (e.g., IDLE, VS Code, or an online interpreter). Ensure proper indentation (4 spaces), handle input errors (e.g., division by zero, invalid input types), and use descriptive print statements. Submit your code files and outputs for review. If you encounter errors, note them and attempt to fix them to deepen your understanding.

1 Assignments

1. **Assignment Arithmetic Operations Calculator: Write a Python program that takes two numbers (integers or floats) as input using `input()`. Perform all arithmetic operations (+, -, *, /, //, %, **), and print each result with a clear label (e.g., "Sum: 8"). Check for division by zero before performing division, floor division, or modulus, and print an error message if the second number is zero.:** Arithmetic Operations Calculator: Write a Python program that takes two numbers (integers or floats) as input using `input()`. Perform all arithmetic operations (+, -, *, /, //, %, **), and print each result with a clear label (e.g., "Sum: 8"). Check for division by zero before performing division, floor division, or modulus, and print an error message if the second number is zero.

2. **Assignment Arithmetic Operator Precedence: Write a program that evaluates the following expressions with different operator precedence and prints the results with explanations in comments. Use variables and `input()` for numbers where appropriate. Test each:**

   - **5 + 4 * 2**
   - **(5 + 4) * 2**
   - **3 ** 2 + 1**
   - **10 / 5 * 2**
   - **10 % 3 // 2**

   Explain in comments why the results differ based on precedence.: Arithmetic Operator Precedence: Write a program that evaluates the following expressions with different operator precedence and prints the results with explanations in comments. Use variables and `input()` for numbers where appropriate. Test each:

   - 5 + 4 * 2

- (5 + 4) * 2
- 3 ** 2 + 1
- 10 / 5 * 2
- 10 % 3 // 2

Explain in comments why the results differ based on precedence.

3. **Assignment Data Type Compatibility: Write a program to test the following statements about arithmetic operators and data types using try-except blocks. For each, attempt the operation, catch any errors, and print whether the statement is True or False with a brief explanation:**

   - **Addition (+) works on strings (e.g., "5" + "3").**
   - **Multiplication (*) works on strings and integers (e.g., "hello" * 3).**
   - **Division (/) works on strings (e.g., "10" / "2").**
   - **Addition (+) works on integers and floats (e.g., 5 + 3.2).**
   - **Exponentiation (**) works on floats (e.g., 2.5 ** 2).**

   : Data Type Compatibility: Write a program to test the following statements about arithmetic operators and data types using try-except blocks. For each, attempt the operation, catch any errors, and print whether the statement is True or False with a brief explanation:

   - Addition (+) works on strings (e.g., "5" + "3").
   - Multiplication (*) works on strings and integers (e.g., "hello" * 3).
   - Division (/) works on strings (e.g., "10" / "2").
   - Addition (+) works on integers and floats (e.g., 5 + 3.2).
   - Exponentiation (**) works on floats (e.g., 2.5 ** 2).

4. **Assignment Relational and Logical Operators: Write a program that takes a number as input and uses relational and logical operators to check the following conditions. Print True or False for each with a descriptive message:**

   - **Is the number between 1 and 10 (inclusive)?**
   - **Is the number positive and even?**
   - **Is the number not equal to zero or greater than 100?**

   Use try-except to handle invalid inputs.: Relational and Logical Operators: Write a program that takes a number as input and uses relational and logical operators to check the following conditions. Print True or False for each with a descriptive message:

   - Is the number between 1 and 10 (inclusive)?
   - Is the number positive and even?
   - Is the number not equal to zero or greater than 100?

   Use try-except to handle invalid inputs.

5. **Assignment Simple Branching with if: Write a program that takes a number as input and uses an `if` statement to check if it is positive. If true, print "The number is positive." Ensure proper indentation (4 spaces). Test with positive,**

negative, and zero inputs.: Simple Branching with if: Write a program that takes a number as input and uses an `if` statement to check if it is positive. If true, print "The number is positive." Ensure proper indentation (4 spaces). Test with positive, negative, and zero inputs.

6. **Assignment if-else Voting Eligibility: Write a program that takes a persons age as input and uses an `if-else` statement to check if they are eligible to vote (age $\geq$ 18). Print "You are eligible to vote." or "You are not eligible to vote." Use try-except for invalid inputs and ensure correct indentation.:** if-else Voting Eligibility: Write a program that takes a persons age as input and uses an `if-else` statement to check if they are eligible to vote (age $\geq$ 18). Print "You are eligible to vote." or "You are not eligible to vote." Use try-except for invalid inputs and ensure correct indentation.

7. **Assignment if-elif-else Grading System: Write a program that takes a score (0100) as input and uses an `if-elif-else` structure to assign a grade: A (90+), B (8089), C (7079), or Fail (below 70). Print the grade with a message (e.g., "Grade: A"). Use try-except for invalid inputs and ensure proper indentation.:** if-elif-else Grading System: Write a program that takes a score (0100) as input and uses an `if-elif-else` structure to assign a grade: A (90+), B (8089), C (7079), or Fail (below 70). Print the grade with a message (e.g., "Grade: A"). Use try-except for invalid inputs and ensure proper indentation.

8. **Assignment Rectangle Calculations: Write a sequential program that takes the length and width of a rectangle as input. Calculate and print the area (length * width) and perimeter (2 * (length + width)). Use try-except to handle invalid inputs (e.g., non-numeric values).:** Rectangle Calculations: Write a sequential program that takes the length and width of a rectangle as input. Calculate and print the area (length * width) and perimeter (2 * (length + width)). Use try-except to handle invalid inputs (e.g., non-numeric values).

9. **Assignment Simple Interest Calculator: Write a sequential program that takes the principal amount, annual interest rate (as a decimal), and time (in years) as input. Calculate the simple interest using the formula (principal * rate * time) / 100, and print the result. Ensure proper input validation with try-except.:** Simple Interest Calculator: Write a sequential program that takes the principal amount, annual interest rate (as a decimal), and time (in years) as input. Calculate the simple interest using the formula (principal * rate * time) / 100, and print the result. Ensure proper input validation with try-except.

10. **Assignment Tax with Dependents: Write a program that combines sequential and branching structures. Take the users annual income and number of dependents as input. Calculate a tax amount (income * 0.15). Use an `if-elif-else` structure to reduce the tax: 20% reduction for 3 or more dependents, 10% reduction for 1 or 2 dependents, no reduction otherwise. Print the income, initial tax, reduction, and final tax. Use try-except and proper indentation.:** Tax with Dependents: Write a program that combines sequential and branching structures. Take the users annual income and number of dependents as input. Calculate a tax amount (income * 0.15). Use an `if-elif-else` structure to reduce the tax: 20% reduction for 3 or more dependents, 10% reduction for 1 or 2 dependents, no reduction otherwise. Print the income, initial tax, reduction, and final tax. Use try-except and proper indentation.

Submission Guidelines

- Save each assignment as a separate Python file (e.g., `assignment1.py`, `assignment2.py`).
- Include comments in your code to explain your logic, especially for operator precedence, try-except blocks, and branching conditions.
- Run each program with at least two different sets of inputs and note the outputs or any errors.
- Submit your code files and a document with the outputs or error messages for each assignment.