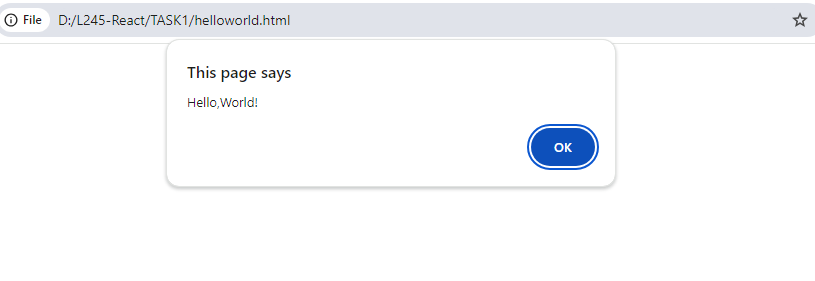
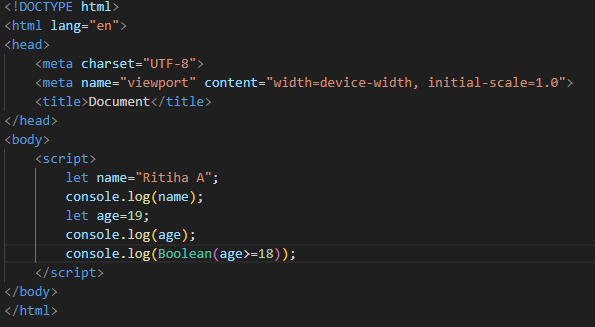
TASK1:

Task 1: Write a simple script that displays “Hello, World!” on the web page using an alert box.

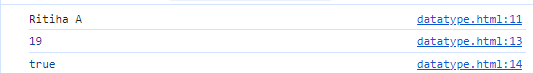
OUTPUT:



Task 2: Experiment with different data types in JavaScript (e.g., string, number, boolean) by declaring and logging them in the console.



Output:

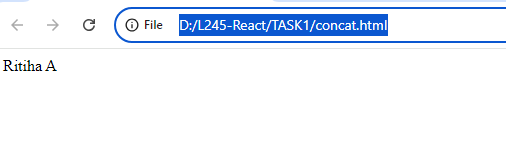


Task 3: Use the console to perform basic math operations like addition, subtraction, multiplication, and division.

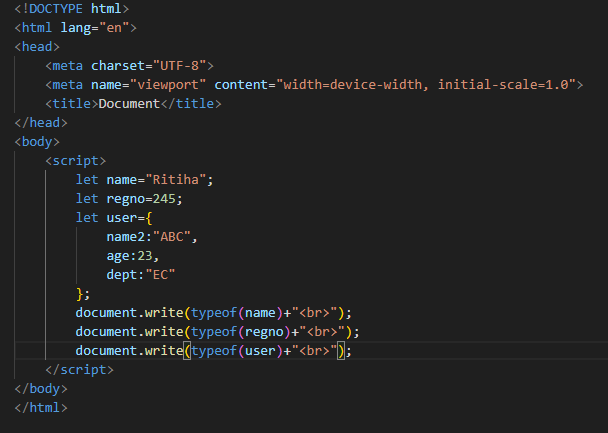


Task 4: Declare two strings and concatenate them using the + operator. 

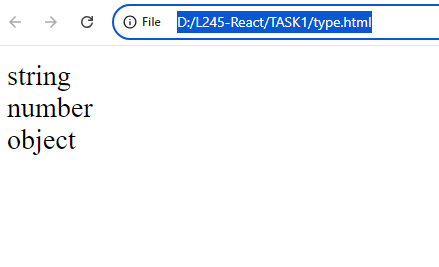
Output:



Task 5: Use the typeof operator to check the data type of various variables.



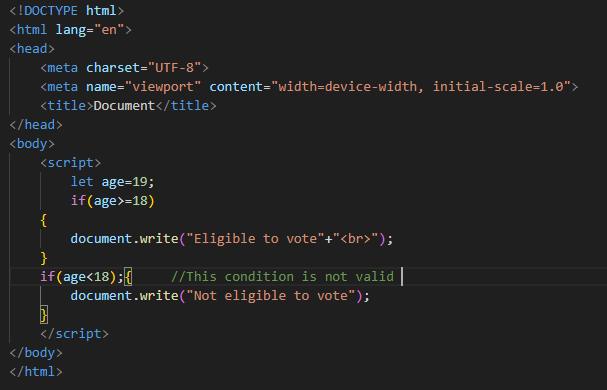
Output:



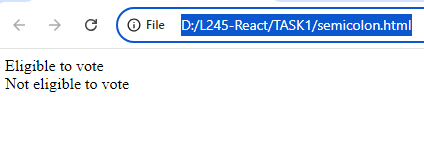
Task 6: Write a multi-line JavaScript comment and a single-line comment. Explain the difference.



Task 7: Create a script with both semicolon-separated and not separated lines. Note any differences in behavior.

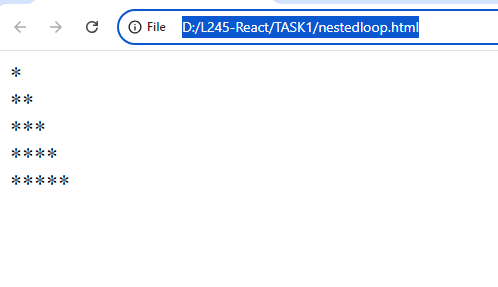


Output:

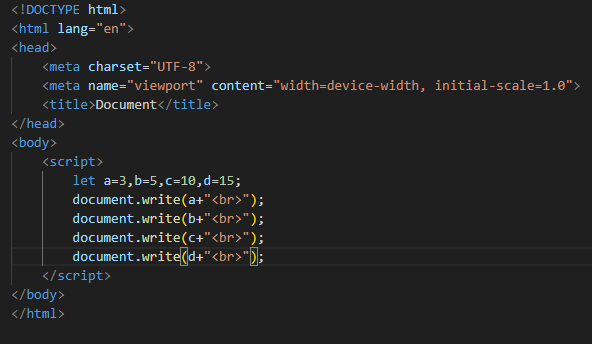


Task 8: Use proper indentation to format a nested loop.

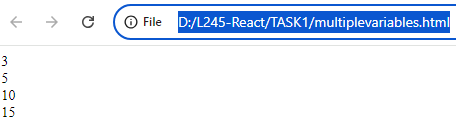
Output:



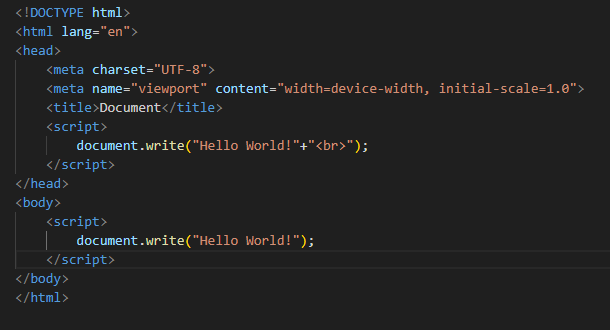
Task 9: Declare multiple variables in a single line.



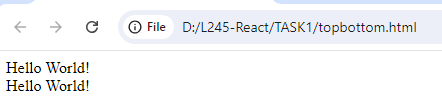
Output:



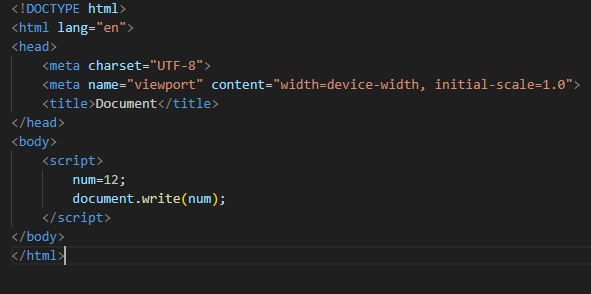
Task 10: Place a script tag at the top and bottom of an HTML document. Note any differences in behavior.



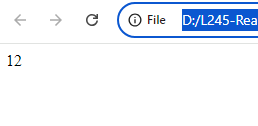
Output:



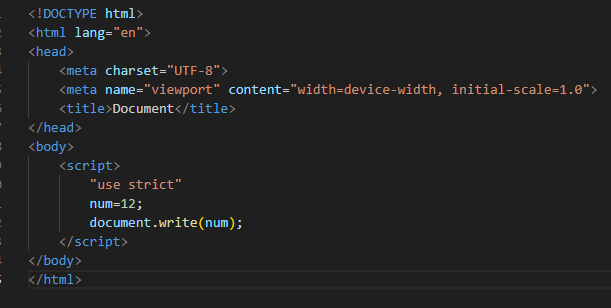
Task 11: Write a script without using “use strict” and try to assign a value to an undeclared variable. Note the result.



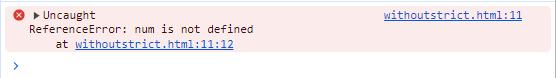
Output:



Task 12: Enable “use strict” mode and repeat the above action, noting the difference.



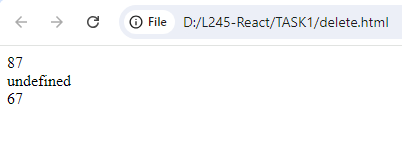
Output:



Task 13: In “use strict” mode, try to delete a variable, function, or function parameter.



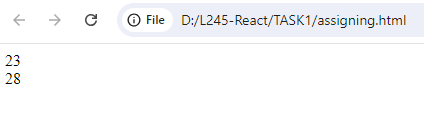
Output:



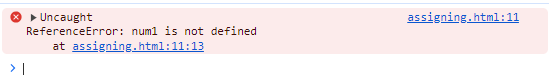
Task 14: Assign a value to an undeclared variable without “use strict” and then with “use strict”.



Output:



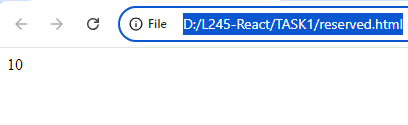




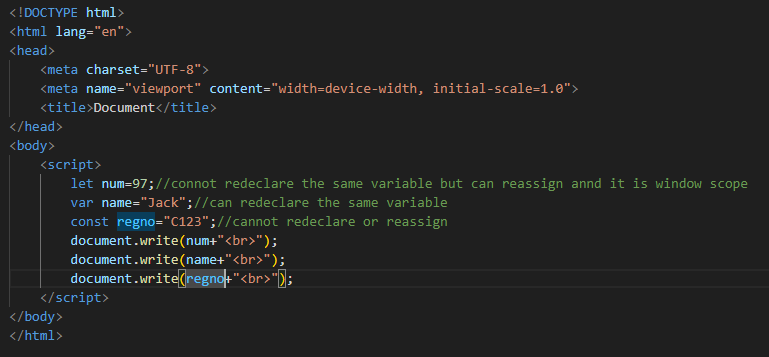
Task 15: Declare a variable with a reserved keyword in “use strict” mode.



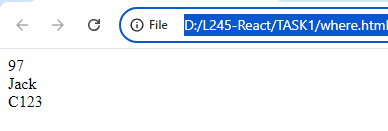
Output:



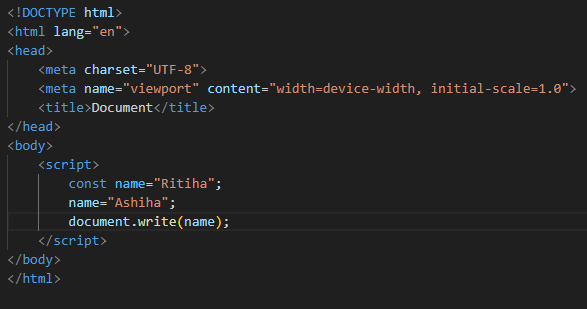
Task 16: Declare variables using let, const, and var. Discuss when each should be used.



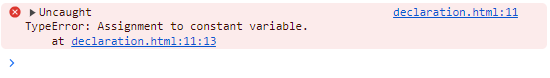
Output:



Task 17: Attempt to reassign a const variable and observe the result.



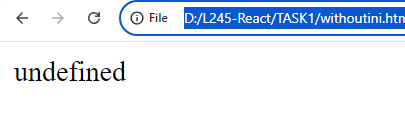
Output:



Task 18: Declare a variable without initializing it and print its value.



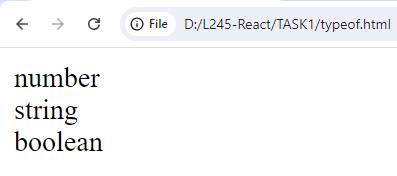
Output:



Task 19: Assign a number, string, and boolean value to a variable and print its type using typeof.



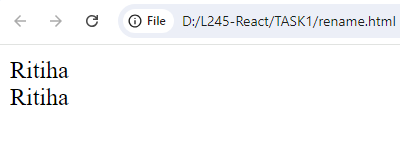
Output:



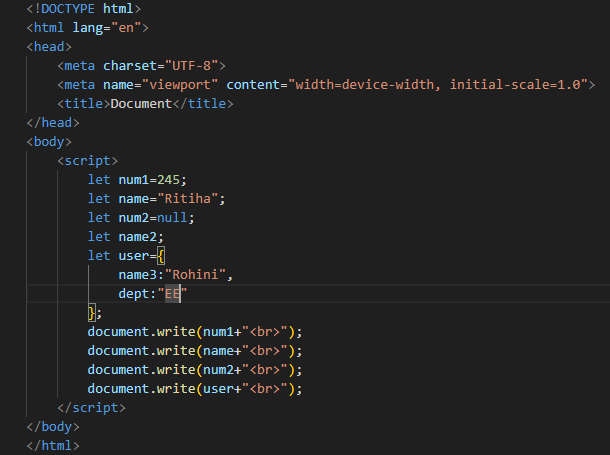
Task 20: Rename a variable and observe the outcome.



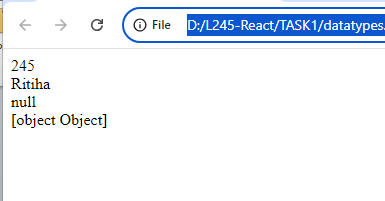
Output:



Task 21: Create variables of different data types (e.g., string, number, boolean, null, undefined, object).



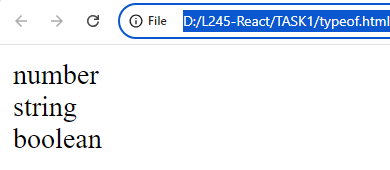
Output:



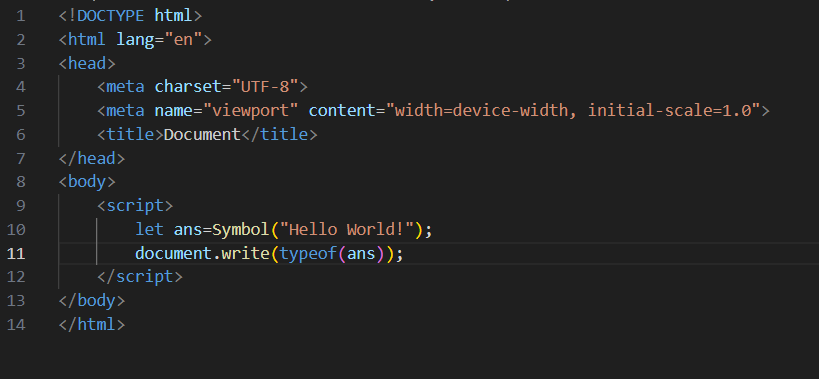
Task 22: Use the typeof operator to determine the type of various variables.



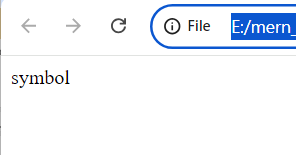
Output:



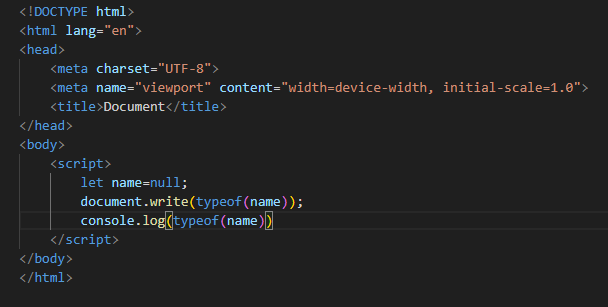
Task 23: Declare a symbol and print its type.



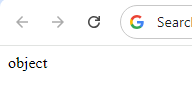
Output:



Task 24: Assign the value null to a variable and check its type using typeof.

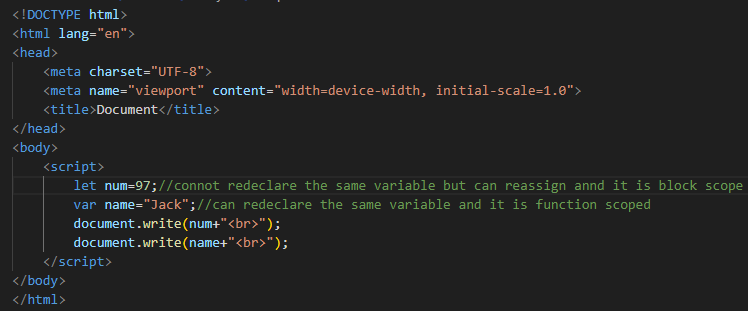


Output:

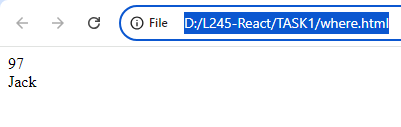




Task 25: Differentiate between declaring a variable using var and let in terms of scope.



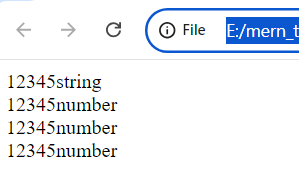
Output:



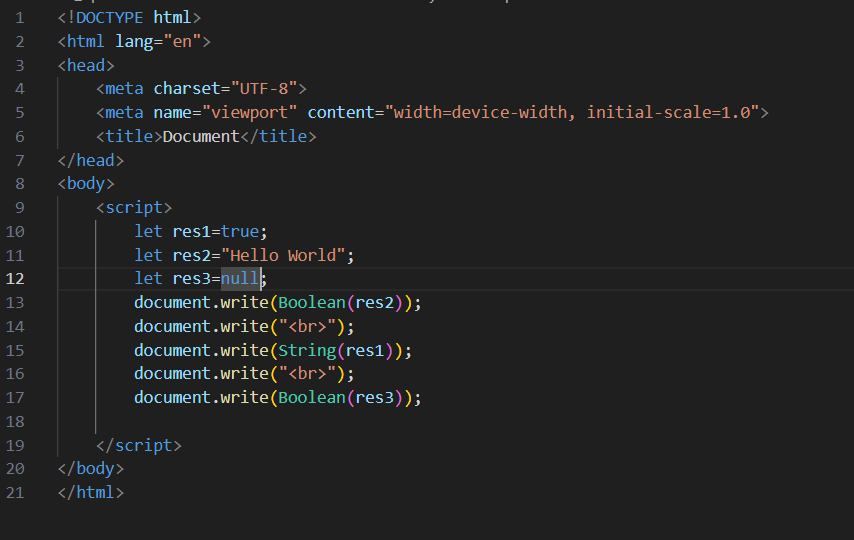
Task 26: Convert a string to a number using both implicit and explicit conversion.



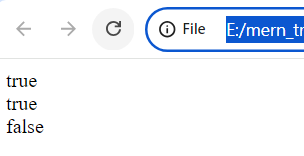
Output:



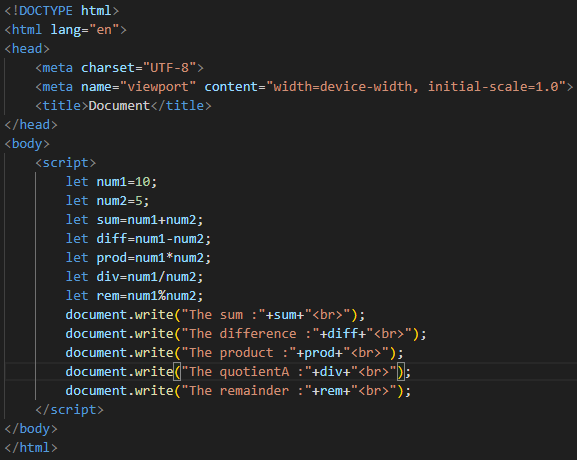
Task 27: Convert a boolean to a string and vice versa.



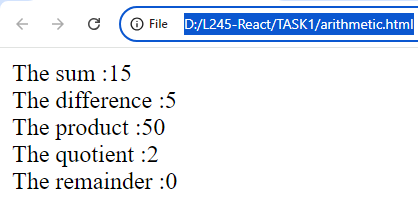
Output:



Task 28: Practice basic arithmetic operators (+, -, \*, /, %).



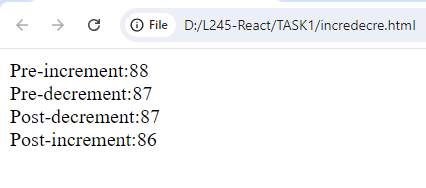
Output:



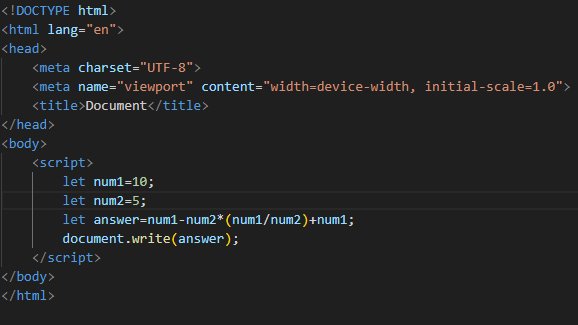
Task 29: Use the ++ and -- operators on a numeric variable.



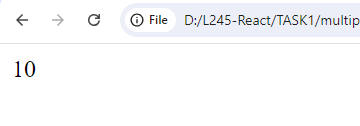
Output:



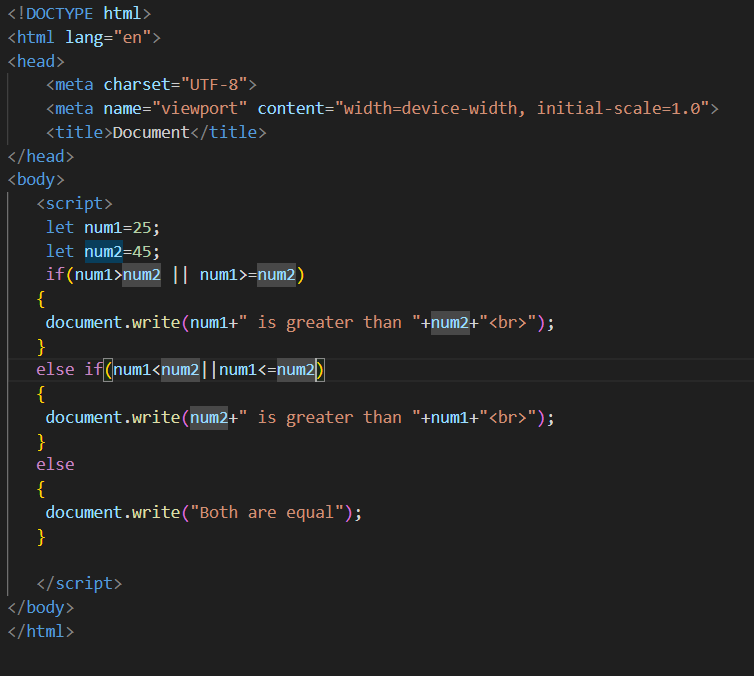
Task 30: Explore the precedence of operators by combining multiple operators in a single expression.



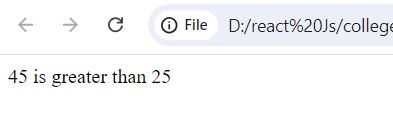
Output:



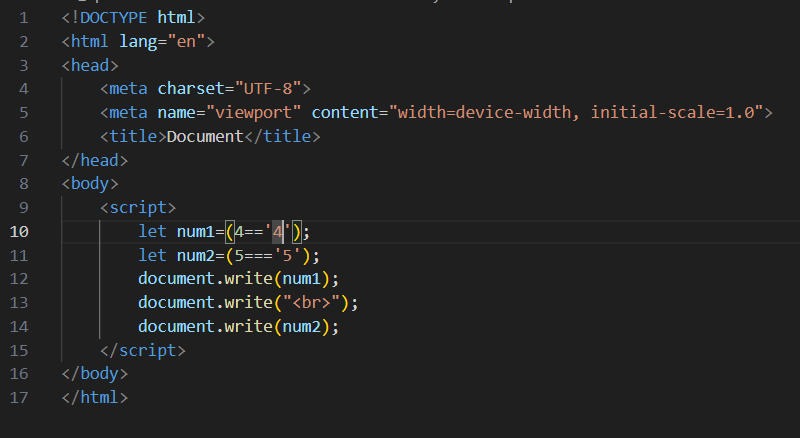
Task 31: Compare two numbers using relational operators (>, =, <=).



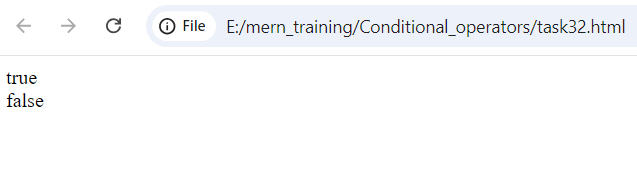
Output:



Task 32: Use equality () and strict equality (=) operators to compare different data types and note the differences.



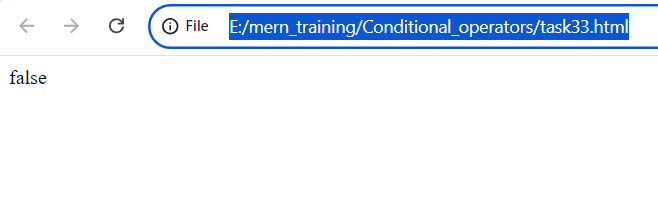
Output:



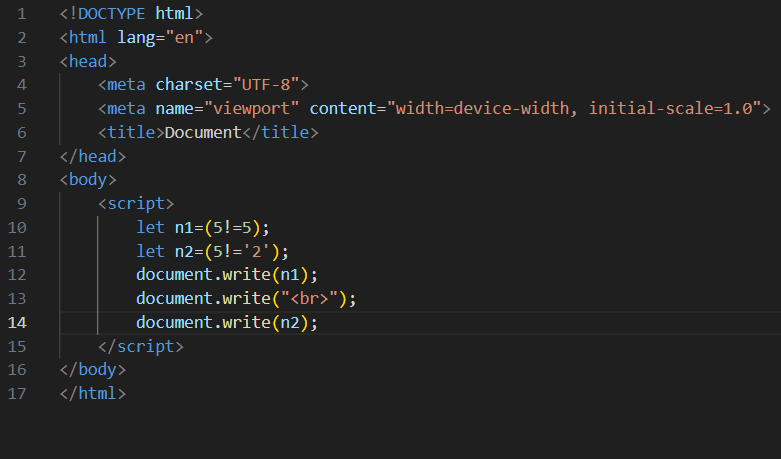
Task 33: Compare two strings lexicographically.



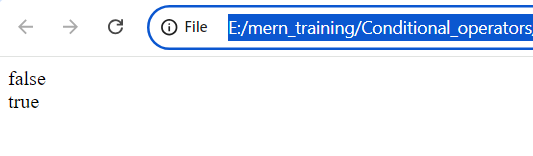
Output:



Task 34: Use the inequality (!=) and strict inequality (!==) operators to compare values.



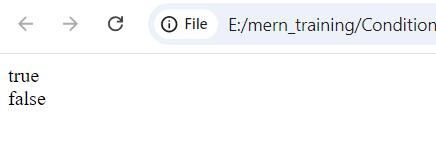
Output:



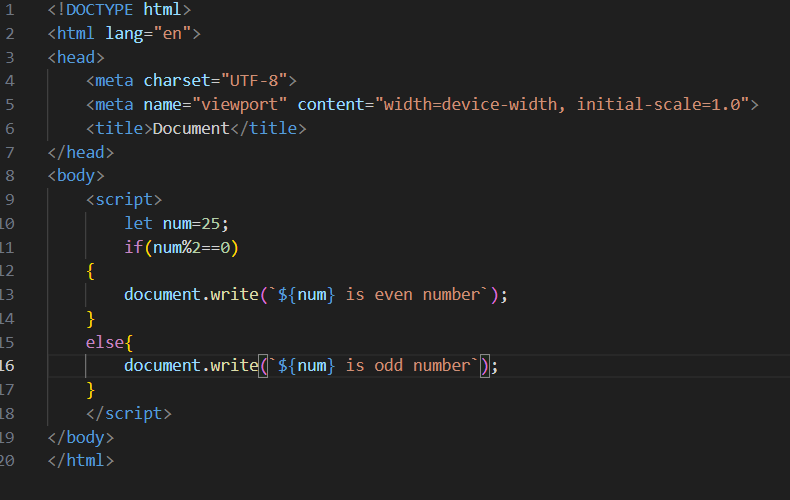
Task 35: Compare null and undefined using both == and ===.



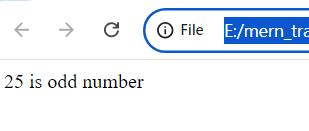
Output:



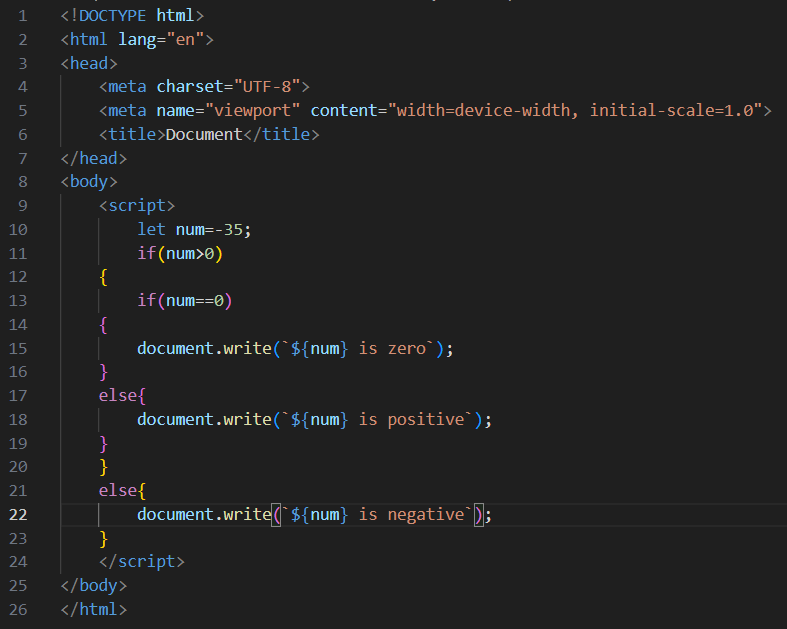
Task 36: Write an if statement that checks if a number is even or odd.



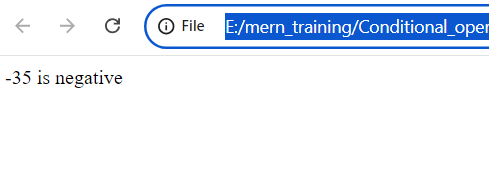
Output:



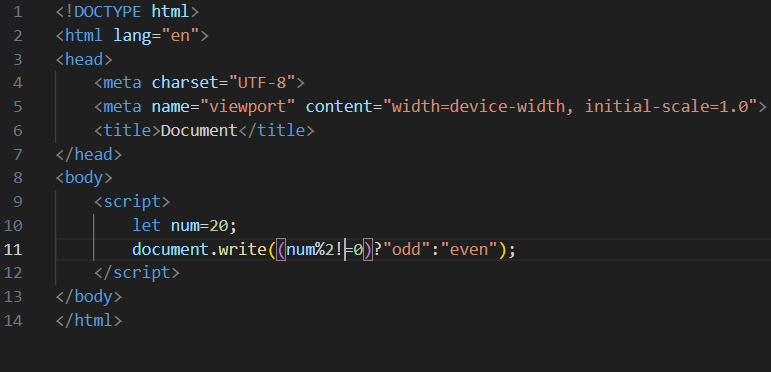
Task 37: Use nested if statements to classify a number as negative, positive, or zero.



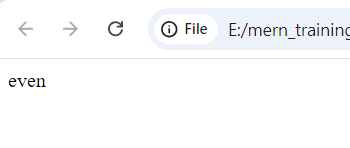
Output:



Task 38: Use the conditional (ternary) operator ‘?’ to rewrite a simple if…else statement.



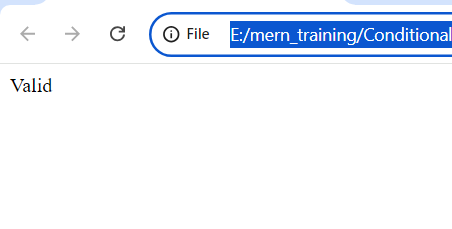
Output:



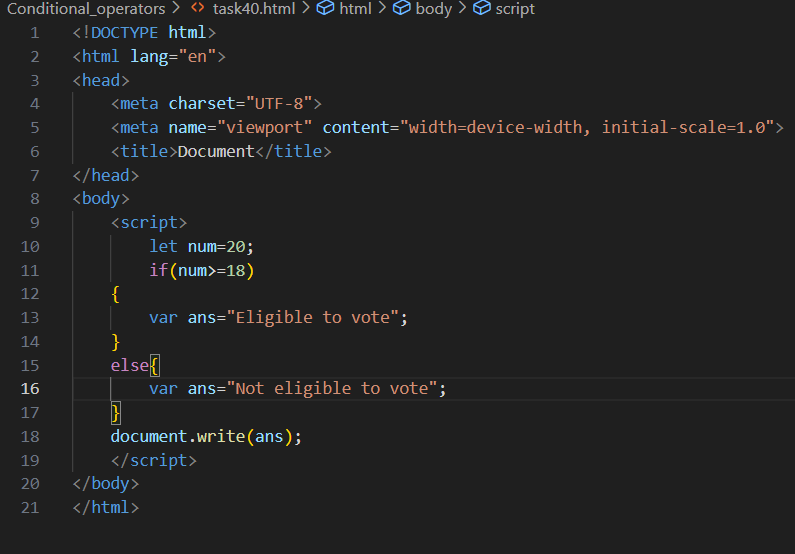
Task 39: Check the validity of a variable using the ? operator.



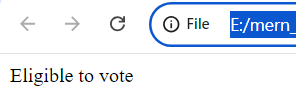
Output:



Task 40: Use the conditional operator to assign a value to a variable based on a condition.



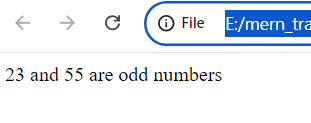
Output:



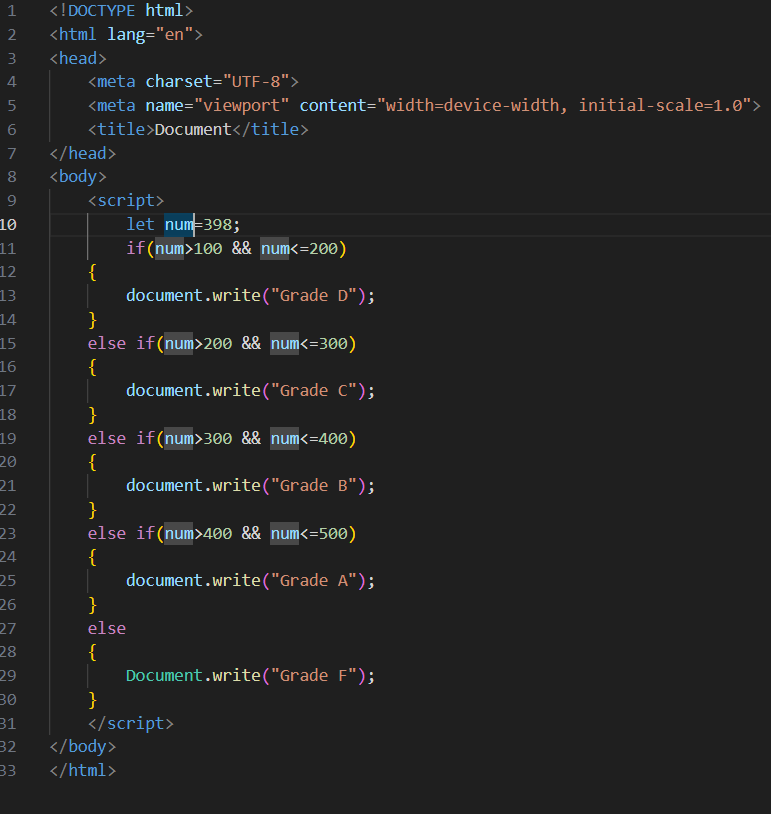
Task 41: Evaluate various combinations of logical operators (&&, ||, !).



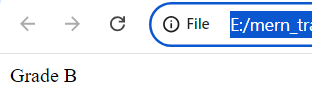
Output:



Task 42: Use logical operators to write a condition that checks if a number is in a given range.



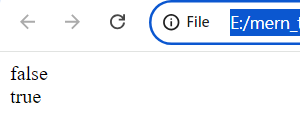
Output:



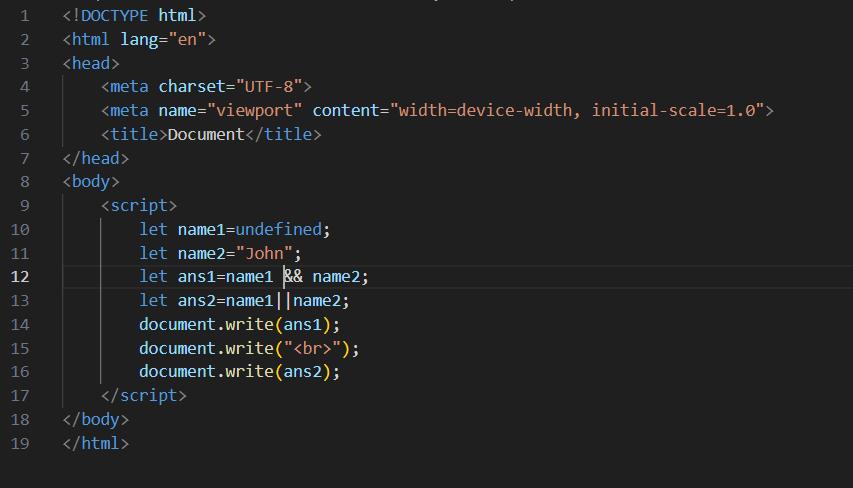
Task 43: Use the NOT (!) operator to invert a boolean value.



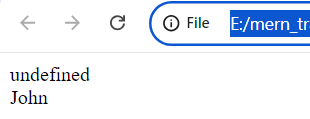
Output:



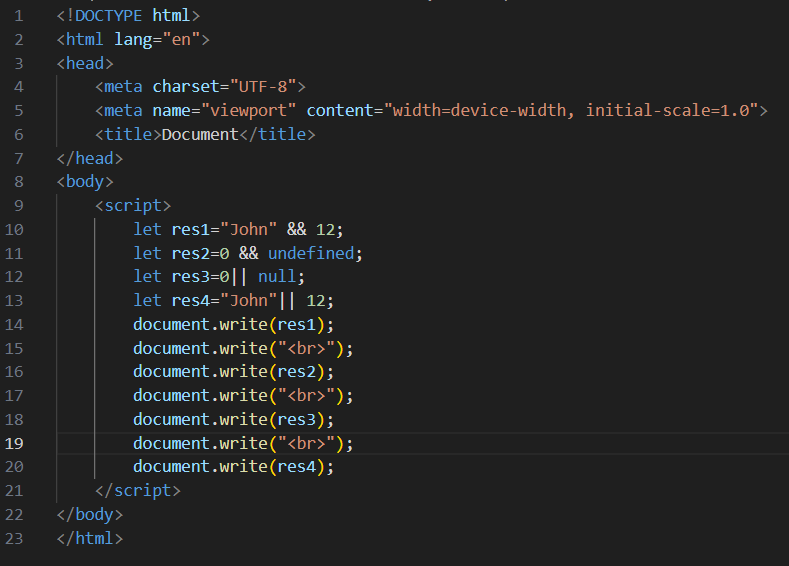
Task 44: Evaluate the short-circuiting nature of logical operators.



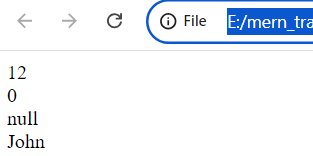
Output:



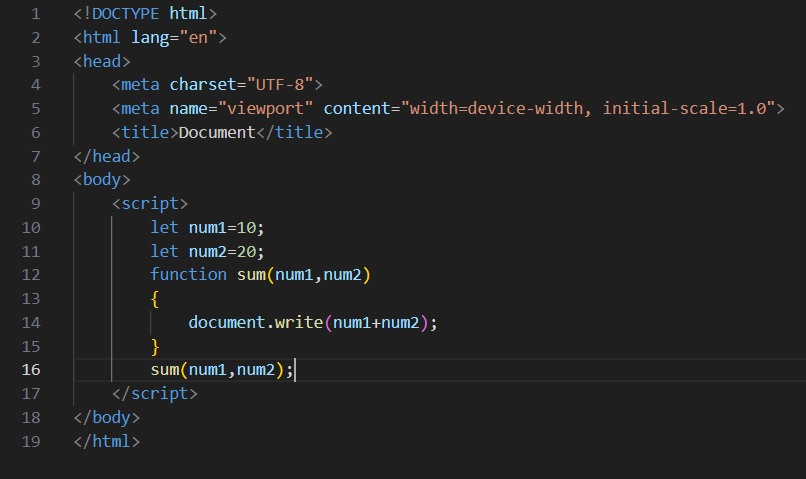
Task 45: Compare two non-boolean values using logical operators and observe the result.



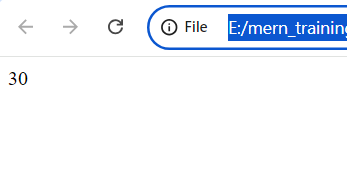
Output:



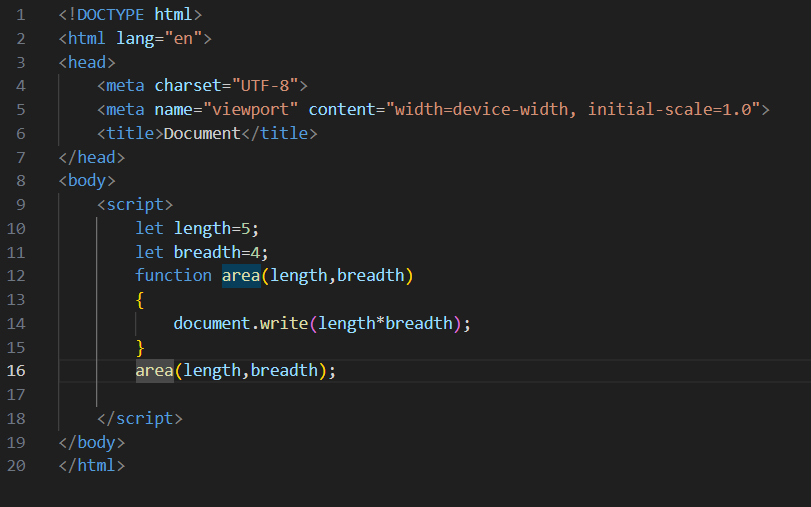
Task 46: Write a function that takes two numbers as arguments and returns their sum.



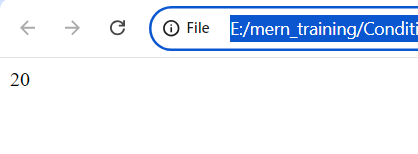
Output:



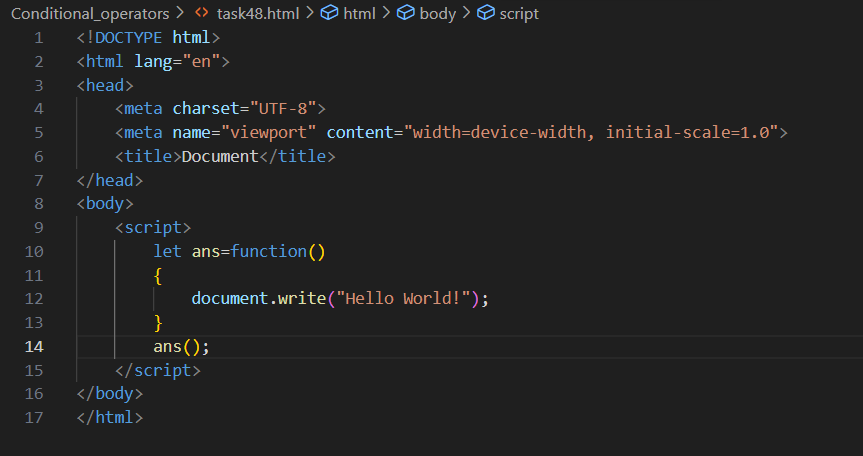
Task 47: Create a function that calculates the area of a rectangle.



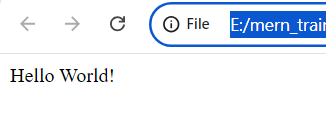
Output:



Task 48: Declare a function without parameters and call it.



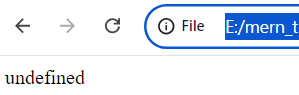
Output:



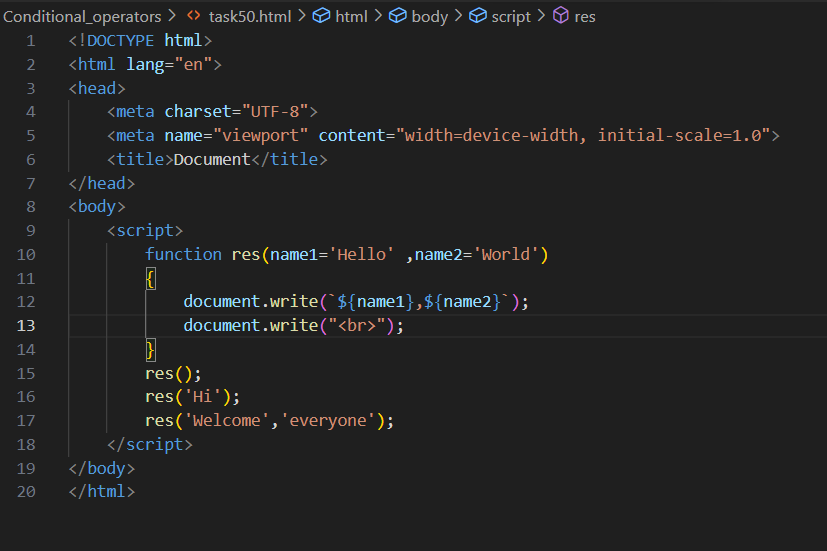
Task 49: Write a function that returns nothing and observe the default return value.



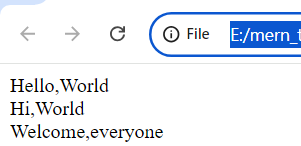
Output:



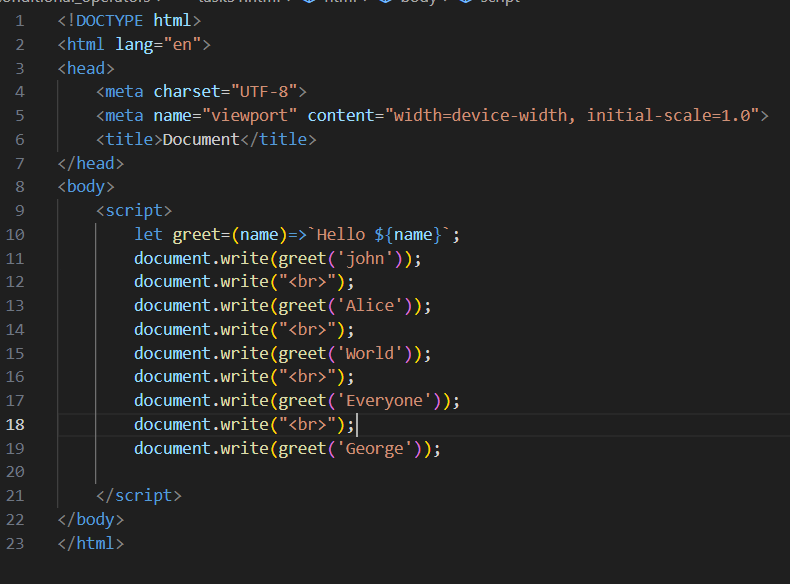
Task 50: Declare a function with default parameters and call it with different arguments.



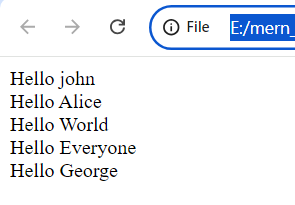
Output:



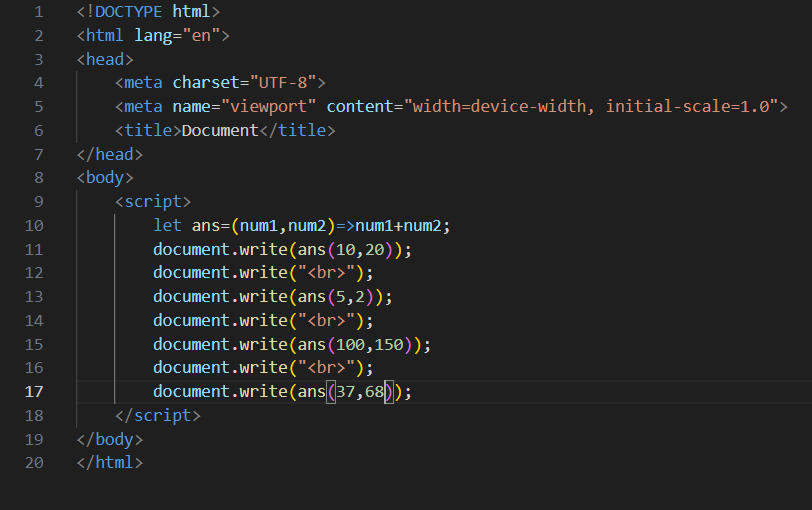
Task 51: Declare a simple arrow function named greet that takes one parameter name and returns the string “Hello, name!”. Test your function with various names.



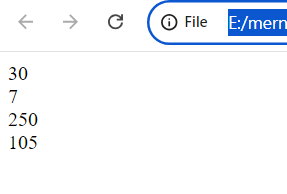
Output:



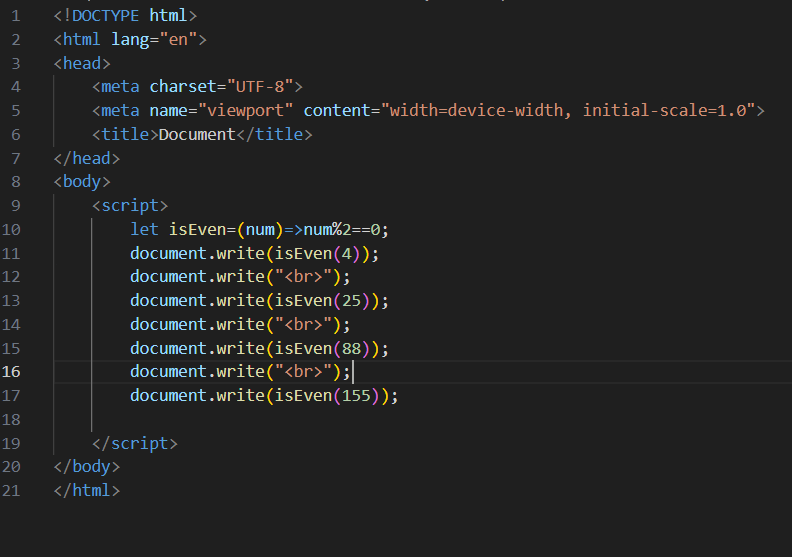
Task 52: Write an arrow function named add that takes two parameters and returns their sum. Validate your function with several pairs of numbers.



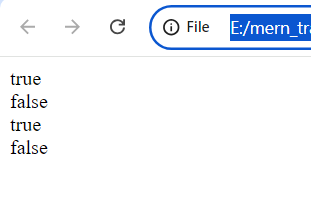
Output:



Task 53: Declare an arrow function named isEven that checks if a number is even. If the number is even, it should return true; otherwise, false. Remember that if the arrow function body has a single statement, you can omit the curly braces.



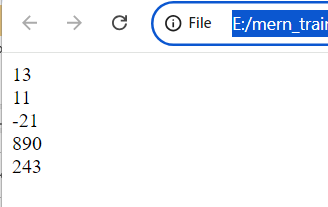
Output:



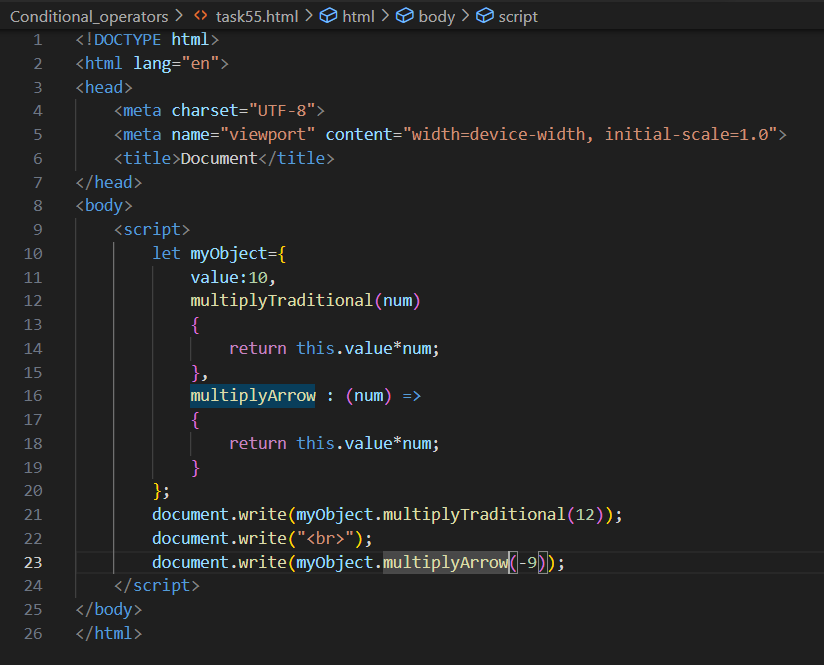
Task 54: Implement an arrow function named maxValue that takes two numbers as parameters and returns the larger number. Here, you'll need to use curly braces for the function body and the return statement.



Output:



Task 55: Examine the behavior of the this keyword inside an arrow function vs a traditional function. Create an object named myObject with a property value set to 10 and two methods: multiplyTraditional using a traditional function and multiplyArrow using an arrow function. Both methods should attempt to multiply the value property by a number passed as a parameter. Check the value of this inside both methods.



Output:

