**Content:**

**JavaScript Scope, Closure and Hoisting:**

* JavaScript Scope
* Global Scope
* Local Scope
* Closure
* Hoisting

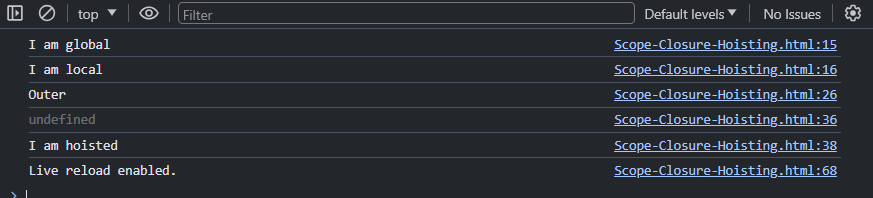
**ES6: Object-Oriented JavaScript:**

* Objects
* Object Properties
* Class Members and Methods
* Prototype and Class
* Inheritance

**JavaScript Scope, Closure, and Hoisting:**

* **JavaScript Scope:**
  + **Scope** in JavaScript refers to the accessibility or visibility of variables in different parts of your code.
  + **Global Scope:** Variables declared outside of any function are accessible from anywhere within your code. They have global scope.
  + **Local Scope:** Variables declared inside a function are accessible only within that function. They have local scope and cannot be accessed from outside the function.
* **Closure:**
  + **Closure** is a powerful feature in JavaScript that allows a function to remember and access its lexical scope even when it's executed outside that scope.
  + It occurs when a function is defined within another function, and the inner function has access to the outer function's variables and parameters even after the outer function has finished executing.
* **Hoisting:**
  + **Hoisting** is a JavaScript mechanism where variables and function declarations are moved to the top of their containing scope during the compilation phase.
  + Only the declarations are hoisted, not the initializations. This means that variables are initialized with a value of undefined until their assignment is encountered during execution.
  + Function declarations are hoisted before variable declarations. This allows you to call functions before they appear in the code.





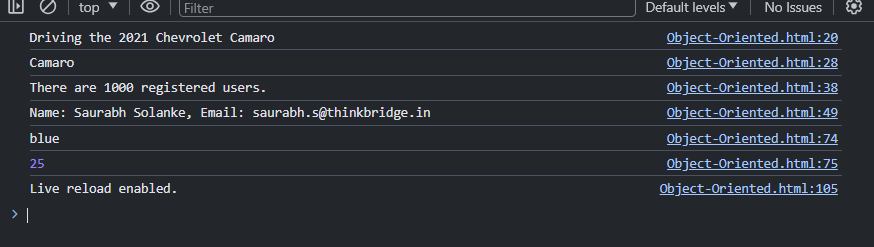
Explanation:

* **Scope**:
  + globalVariable is accessible everywhere in the script.
  + localVariable is only accessible within myFunction.
* **Closure**:
  + innerFunction can still access outerVariable even after outerFunction finishes executing.
* **Hoisting**:
  + Variable declarations with var are moved to the top of their scope. Assignments remain in place.

**ES6 : Object-Oriented JavaScript:**

* **Objects:**
  + In JavaScript, an **object** is a collection of key-value pairs where keys are strings (or Symbols) and values can be any data type, including other objects, functions, or primitive values like strings, numbers, or booleans.
  + Objects allow you to group related data and functionality together, making your code more organized and easier to manage.
* **Object Properties:**
  + **Object properties** are the attributes or characteristics of an object that hold data.
  + They can be accessed and modified using either dot notation (object.property) or bracket notation (object['property']).
* **Class Members and Methods:**
  + **Class members** are properties and methods defined within a class.
  + **Methods** are functions defined within a class that perform some action or return a value associated with the object created from that class.
* **Prototype and Class:**
  + **Prototype** is an object from which other objects inherit properties and methods.
  + In JavaScript, objects can inherit properties and methods from other objects through a mechanism called **prototypal inheritance**.
  + **Class** is a new syntax introduced in ECMAScript 2015 (ES6) for creating objects and implementing inheritance in a more structured and familiar way, similar to other object-oriented programming languages like Java or C++.
* **Inheritance:**
  + **Inheritance** is a mechanism in which a class inherits properties and behaviors from another class.
  + It allows you to create new classes that reuse and extend the functionality of existing classes, promoting code reusability and modularity.



  
  
Explanation:

* **Objects**: car object contains details about a car. The drive method allows it to be driven.
* **Object Properties**: Accesses an object property using bracket notation with a variable.
* **Class Members and Methods**: Defines a User class with a static method countUsers and an instance method getInfo. Static methods are called on the class itself, while instance methods are called on instances of the class.
* **Prototype and Class**: Demonstrates class inheritance. Square inherits from Shape. The super keyword calls the constructor of the parent class.