# SHORT CIRCUITING

Short-circuiting in JavaScript is an evaluation strategy used with logical operators (`&&` (AND), `||` (OR), and `??` (Nullish Coalescing)) where the evaluation of an expression stops as soon as the result can be determined. This means that not all parts of the expression are necessarily evaluated, potentially leading to performance improvements and more concise code.

Here's how it works with each operator:

- Logical AND (`&&`): The `&&` operator evaluates from left to right. If the left operand is `false` (or a falsy value like `0`, `null`, `undefined`, `NaN`, `""`), the expression immediately short-circuits and returns that falsy value, without evaluating the right operand. This is because if the first part of an AND condition is false, the entire condition must be false.

  JavaScript
  ```javascript
  const result = false && doSomething(); // doSomething() is never called
  ```

- Logical OR (`||`): The `||` operator also evaluates from left to right. If the left operand is `true` (or a truthy value), the expression immediately short-circuits and returns that truthy value, without evaluating the right operand. This is because if the first part of an OR condition is true, the entire condition must be true.

  JavaScript
  ```javascript
  const result = true || doSomething(); // doSomething() is never called
  ```

- Nullish Coalescing (`??`): The `??` operator is similar to `||` but specifically checks for `null` or `undefined`. If the left operand is `null` or `undefined`, it evaluates and returns the right operand; otherwise, it returns the left operand without evaluating the right.

  JavaScript
  ```javascript
  const value = null ?? "default"; // value is "default"
  const anotherValue = 0 ?? "default"; // anotherValue is 0 (0 is not null or
  ```

Benefits of Short-Circuiting:

- **Performance Optimization**:

  Avoids unnecessary computations when the result of a logical expression is already determined.

- **Concise Code**:

  Allows for more compact and readable code, often replacing `if/else` statements for simple conditional assignments or function calls.

- **Safe Property Access**:
  Useful for safely accessing nested object properties, preventing errors if an intermediate property is `null` or `undefined`.