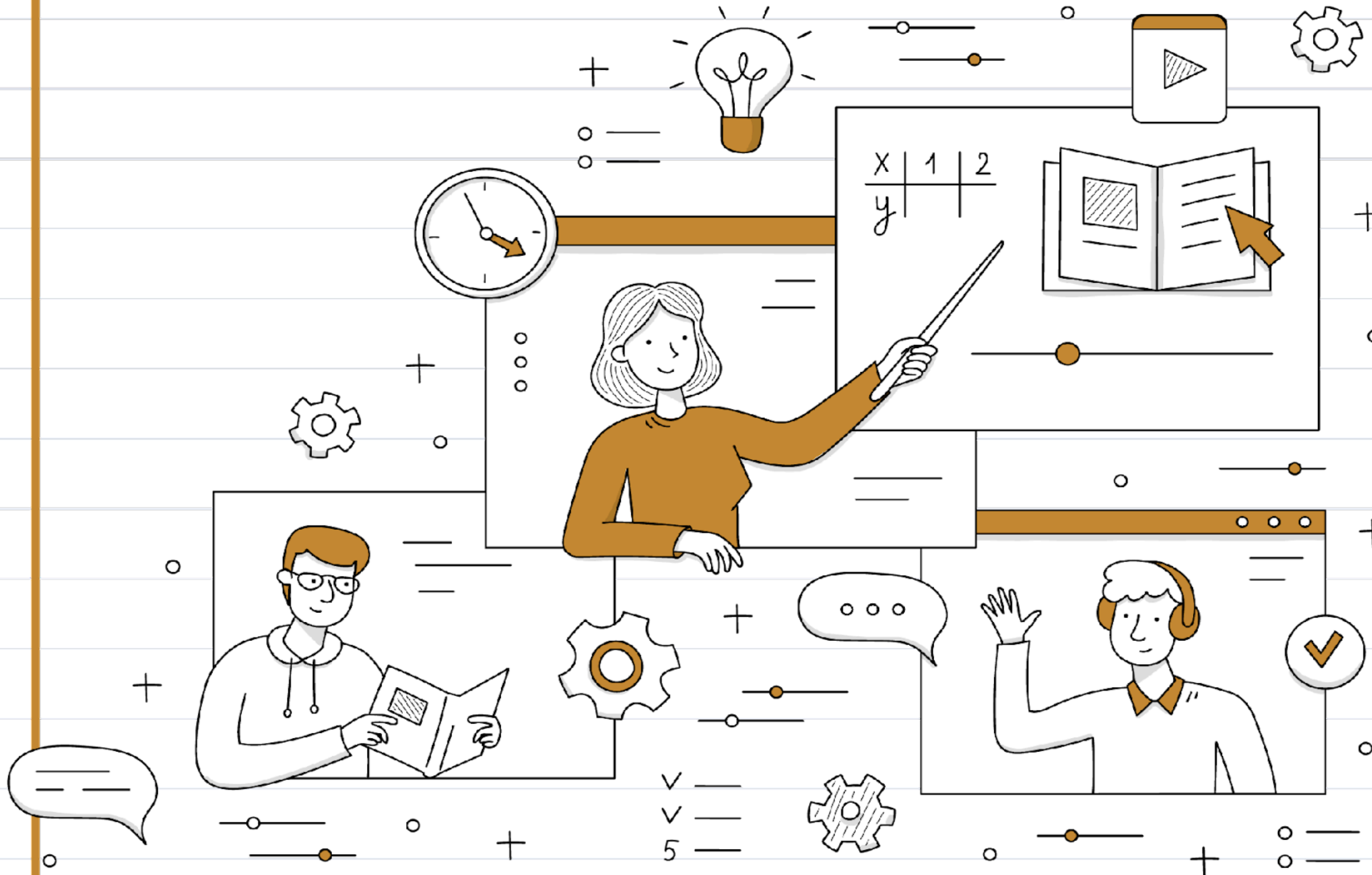
SUNSTONE

JavaScript

LECTURE 4 NOTES

# Expressions and Operators



- **What are Expressions?**

  - In JavaScript, an expression is a combination of values, variables, operators, and function calls that evaluates a result. Expressions can be as simple as a single value or as complex as a combination of multiple operations.
  - An expression is a block of code that evaluates to a value. JavaScript's expression is a valid set of literals, variables, operators, and expressions that evaluate a single value that is an expression. This single value can be a number, a string, or a logical value depending on the expression.
  - **For example,** this is an expression `2*(4-1)`

- **What are Operators?**

  - In JavaScript, operators are symbols or special keywords that perform operations on values, such as arithmetic operations, logical comparisons, assignments, and more. They are used to manipulate and combine values in expressions to produce a desired result.
  - JavaScript operators operate the operands, these are symbols that are used to manipulate a certain value or operand. Operators are used to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. In JavaScript, operators are used to compare values, perform arithmetic operations, etc.
  - For example,

    ```
    x=y+z
    ```

    The Addition Operator + adds numbers:

    The Assignment Operator = assigns a value to a variable.

    x, y, & z are Operands.

  - **Note:** Expressions are used to calculate a result involving a number of operands (either variables or constants). Operators allow us to manipulate the variables and constants in the expressions. Operands are the constants or variables which the operators operate upon.
  - Here are some examples of expressions in JavaScript:

- **Arithmetic Expressions:**
```
5 + 3          // Addition expression
2 * (4 - 1)    // Subtraction and multiplication expression
```
- **Variable Expressions:**
```
var x = 10;
x + 5                    // Variable and addition expression
```
- **String Expressions:**
```
"Hello" + " " + "World"
//Concatenation of strings expression
```
- **Function Call Expressions:**
```
Math.sqrt(25)            // Calling a function expression
```
- **Logical Expressions:**
```
(x > 5) && (x < 10)      // Logical AND expression
```
- **Conditional (Ternary) Expressions:**
```
var result = (x > 0) ? "Positive" : "Negative";
// Ternary expression
```

JavaScript operators can be classified into several types:

1. **Arithmetic Operators:**
    - **Addition (+):** Adds two values together.
    - **Subtraction (-):** Subtracts the right-hand operand from the left-hand operand.
    - **Multiplication (*):** Multiplies two values.
    - **Division (/):** Divides the left-hand operand by the right-hand operand.
    - **Modulus (%):** Returns the remainder of the division of two values.
    - **Increment (++) and Decrement (--):** Increase or decrease the value of a variable by 1.

2. **Assignment Operators:**
    - **Assignment (=):** Assigns a value to a variable.
    - **Addition assignment (+=):** Adds the right-hand operand to the left-hand operand and assigns the result to the left-hand operand.
    - **Subtraction assignment (-=):** Subtracts the right-hand operand from the left-hand operand and assigns the result to the left-hand operand.
    - **Multiplication assignment (*=):** Multiplies the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
    - **Division assignment (/=):** Divides the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
    - **Modulus assignment (%=):** Calculates the modulus between the left-hand and right-hand operands and assigns the result to the left-hand operand.

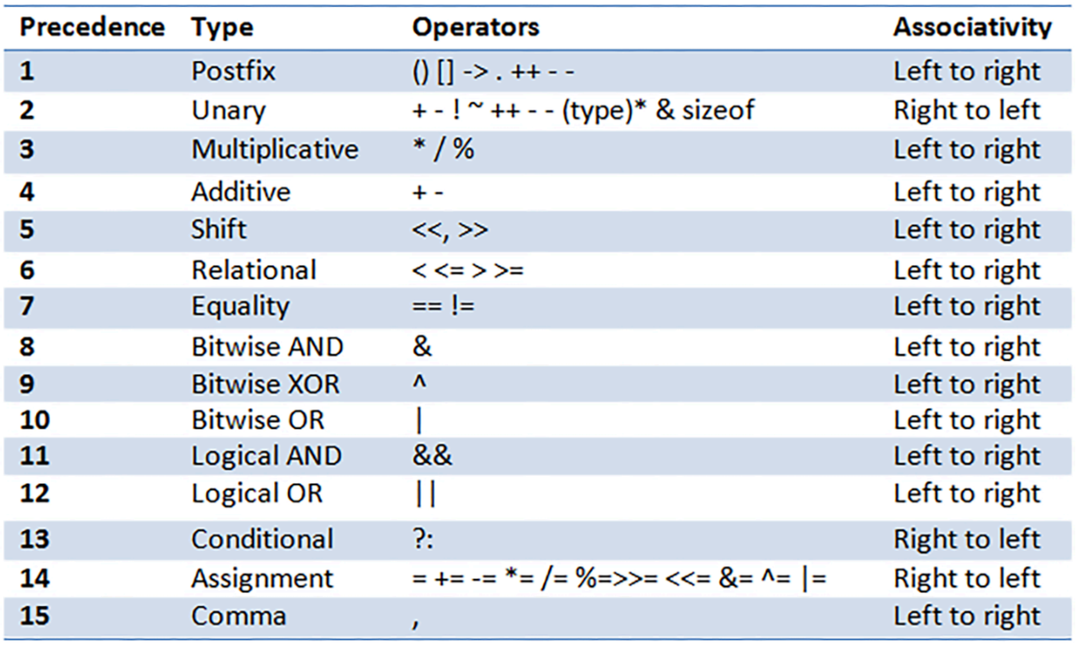
3. **Comparison Operators:**
    - **Equality (==):** Checks if two values are equal, performing type coercion if necessary.
    - **Strict equality (===):** Checks if two values are equal without performing type coercion.
    - **Inequality/Not equal (!=):** Checks if two values are not equal, performing type coercion if necessary.
    - **Strict inequality/Not equal (!==):** Checks if two values are not equal without performing type coercion.
    - **Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=):** Compare the values of two operands.

4. **Logical Operators:**
    - **Logical AND (&&):** Returns true if both operands are true.
    - **Logical OR (||):** Returns true if at least one of the operands is true.
    - **Logical NOT (!):** Negates the truth value of an operand.

5. **Conditional (Ternary) Operator:**
    - **Ternary operator (condition ? expr1 : expr2):** Evaluates a condition and returns one of two expressions based on the result of the condition.

6. **Bitwise Operators:**
    - **Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^), Bitwise NOT (~), Left shift (<<), Right shift (>>), Zero-fill right shift (>>>):** Perform bitwise operations on integer values by manipulating their binary representations.

- **What is Operator Precedence?**
    - In JavaScript, each operator is assigned a precedence level. Operators with higher precedence are evaluated before operators with lower precedence. It refers to the priority given to operators while parsing a statement that has more than one operator performing operations in it. It is important to ensure the correct result and also to help the compiler understand what the order of operations should be. Operators with higher priorities are resolved first. But as one goes down the list, the priority decreases and hence their resolution. It represents how two expressions are bound together. In an expression, it determines the grouping of operators with operands and decides how an expression will evaluate. Precedence is used to tell the compiler what operations should be performed first.
    - **Note:** Operator precedence determines which operator is evaluated first, while associativity comes into play when multiple operators with the same precedence are used.
- **What is Associativity?**
    - If two or more operators have the same precedence, their associativity determines the order of evaluation. Associativity can be either left-to-right (left associative) or right-to-left (right associative). Associativity in general states that irrespective of the order of operands for a given operation the result remains the same.
- **Precedence and Associativity of Operators**
    - The following table describes the precedence and associativity of operators used in JavaScript:



| Precedence | Type | Operators | Associativity |
|---|---|---|---|
| 1 | Postfix | () [] -> . ++ - - | Left to right |
| 2 | Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| 3 | Multiplicative | * / % | Left to right |
| 4 | Additive | + - | Left to right |
| 5 | Shift | <<, >> | Left to right |
| 6 | Relational | < <= > >= | Left to right |
| 7 | Equality | == != | Left to right |
| 8 | Bitwise AND | & | Left to right |
| 9 | Bitwise XOR | ^ | Left to right |
| 10 | Bitwise OR | \| | Left to right |
| 11 | Logical AND | && | Left to right |
| 12 | Logical OR | \|\| | Left to right |
| 13 | Conditional | ?: | Right to left |
| 14 | Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| 15 | Comma | , | Left to right |

- **Expressions and their Evaluation**
    - Let us consider an example of an expression with multiple operators:

```
Result = (4+7*5) -30 >0 ? 1:0;
```

    - The above expression uses arithmetic operators, conditional operators, relational operators, and parentheses.
        - In order to evaluate this expression, first we need to evaluate the expression within the parenthesis.
        - It has arithmetic expressions. Its priorities are first multiplication and then addition. Hence we need to evaluate 7*5 first and add this result with 4. Hence the result within parenthesis is 39.
        - In order to evaluate a conditional operator, we need to complete all the arithmetic operations on LHS. Hence subtract 30 from 39. It results in 9.
        - Now check 9>0. It's correct and returns TRUE. Hence the expression results in 1.
        - In the above expression, we have considered evaluating the condition for the conditional operator because its associativity is from right to left. Hence it needs to be evaluated first which in turn requires its condition to be evaluated.
        - This condition is arithmetic which is evaluated from left to right starting from expression within parenthesis and then subtraction. This is how any expression

with multiple operators is evaluated.

- **Note:** To evaluate the expression, we can also use a pre-defined function `eval()`.

  Syntax is `eval(string);`

  ```
  console.log(eval((4+7*5) -30 >0 ? 1:0)); //Output: 1
  ```