ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Chapter 11 (2)

Operators and Statements

Objectives

- Explain operators and their types in JavaScript
- Explain regular expressions in JavaScript
- Explain decision-making statements in JavaScript

Introduction

An operator specifies the type of operation to be performed on the values of variables and expressions.

JavaScript provides different types of operators to perform simple to complex calculations and evaluations.

Certain operators are also used to construct relational and logical statements. These statements allow implementing decision and looping constructs.

Basics of Operators 1-2

An operation is an action performed on one or more values stored in variables.

The specified action either changes the value of the variable or generates a new value.

An operation requires minimum one symbol and some value.

Symbol is called an operator and it specifies the type of action to be performed on the value.

Value or variable on which the operation is performed is called an operand.

Basics of Operators 2-2

- Three main types of operators are as follows:

Unary operators - Operates on a single operand.

Binary operators - Operates on two operands.

Ternary operators - Operates on three operands.

Operators and their Types

Operators help in simplifying expressions.

JavaScript provides a predefined set of operators that allow performing different operations.

JavaScript operators are classified into six categories based on the type of action they perform on operands.

- Six categories of operators are as follows:
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Assignment operators
  - Bitwise operators
  - Special operators

Arithmetic Operators 1-2

Are binary operators.

Perform basic arithmetic operations on two operands.

Operator appears in between the two operands, which allow you to perform computations on numeric and string values.

- Following table lists arithmetic operators.

Arithmetic Operator	Description	Example
+ (Addition)	Performs addition. In case of string values, it behaves as a string concatenation operator and appends a string at the end of the other	45 + 56
- (Subtraction)	Performs subtraction. If a larger value is subtracted from a smaller value, it returns a negative numeric value	76-78
/ (Division)	Divides the first operand by the second operand and returns the quotient	24 / 8
% (Modulo)	Divides the first operand by the second operand and returns the remainder	90 % 20
* (Multiplication)	Multiplies the two operands	98 * 10

Arithmetic Operators 2-2

- The Code Snippet demonstrates the use of arithmetic operators.

```
<SCRIPT>
  var loanAmount = 34500;
  var interest = 8;
  var interestAmount, totalAmount;
  interestAmount = loanAmount * (interest / 100);
  totalAmount = loanAmount + interestAmount;
  document.write("<B>Total amount to be paid ($):</B>" +
totalAmount + "<BR />");
</SCRIPT>
```

Increment and Decrement Operators 1-2

Increment and decrement operators are unary operators.

Increment operator (++) increases the value by 1, while the decrement operator (--) decreases the value by 1.

These operators can be placed either before or after the operand.

Operator if placed before the operand, expression is called pre-increment or pre-decrement. Operator if placed after the operand, expression is called post-increment or post-decrement.

- Following table lists arithmetic operators.

Expressions	Type	Result
numTwo = ++numOne;	Pre-increment	numTwo = 3
numTwo = numOne++;	Post-increment	numTwo = 2
numTwo = --numOne;	Pre-decrement	numTwo = 1
numTwo = numOne--;	Post-decrement	90 % 20

Increment and Decrement Operators 2-2

- The Code Snippet demonstrates the use of unary operators in JavaScript.

```
<SCRIPT>
  var number = 3;
  alert('Number after increment = ' + ++number);
  alert('Number after decrement = ' + number--);
</SCRIPT>
```

Relational Operators 1-3

Are binary operators that make a comparison between two operands.

After making a comparison, they return a boolean value namely, true or false.

Expression consisting of a relational operator is called as the relational expression or conditional expression.

- Following table lists the relational operators.

Relational Operators	Description	Example
== (Equal)	Verifies whether the two operands are equal	90 == 91
!= (Not Equal)	Verifies whether the two operands are unequal	99 != 98
=== (Strict Equal)	Verifies whether the two operands are equal and are of the same type	numTwo = 1
!== (Strict Not Equal)	Verifies whether the two operands are unequal and whether are not of the same type	90 % 20

Relational Operators 2-3

- Following table lists some more relational operators.

Relational Operators	Description	Example
> (Greater Than)	Verifies whether the left operand is greater than the right operand	97 > 95
< (Less Than)	Verifies whether the left operand is less than the right operand	94 < 96
>= (Greater Than or Equal)	Verifies whether the left operand is greater than or equal to the right operand	92 >= 93
<= (Less Than or Equal)	Verifies whether the left operand is less than or equal to the right operand	99 <= 100

Relational Operators 3-3

- The Code Snippet demonstrates the use of relational operators in JavaScript.

```
<SCRIPT>
  var firstNumber = 3;
  var secondNumber = 4;
  document.write('First number is greater than the second
number: ' + (firstNumber > secondNumber));
  document.write('<br/>First number is less than the
second number: ' + (firstNumber < secondNumber));
  document.write('<br/>First number is equal to the second
number: ' + (firstNumber == secondNumber));
</SCRIPT>
```

Logical Operators 1-2

Are binary operators that perform logical operations on two operands.

They belong to the category of relational operators, as they return a boolean value.

- Following table lists the logical operators.

Logical Operators	Description	Example
&& (AND)	Returns true, if either of the operands are evaluated to true. If first operand evaluates to true, it will ignore the second operand	(x == 2) && (y == 5) Returns false
! (NOT)	Returns false, if the expression is true and vice-versa	!(x == 3) Returns true
|| (OR)	Returns true, if either of the operands are evaluated to true. If first operand evaluates to true, it will ignore the second operand	(x == 2) || (y == 5) Returns true

Logical Operators 2-2

- The Code Snippet demonstrates the use of logical AND operator in JavaScript.

```
<SCRIPT>
  var name = “John”;
  var age = 23;
  alert(‘John\’s age is greater than or equal to 23 years : ‘ +
((name==”John”) && (age >= 23)));
</SCRIPT>
```

Assignment Operators

Assignment operators assign the value of the right side operand to the operand on the left side by using the equal to operator (=).

Simple assignment operator - Is the '=' operator which is used to assign a value or result of an expression to a variable.

Compound assignment operator - Is formed by combining the simple assignment operator with the arithmetic operators.

- Following table lists the assignment operators.

Expressions	Description	Example
numOne += 6;	numOne = numOne + 6	numOne = 12
numOne -= 6;	numOne = numOne – 6	numOne = 0
numOne *= 6;	numOne = numOne * 6	numOne = 36
numOne %= 6;	numOne = numOne % 6	numOne = 0
numOne /= 6;	numOne = numOne / 6	numOne = 1

Bitwise Operators 1-2

Represent their operands in bits (zeros and ones) and perform operations on them.

They return standard decimal values.

Compound assignment operator - Is formed by combining the simple assignment operator with the arithmetic operators.

Bitwise Operators 2-2

- Following table lists the bitwise operators in JavaScript.

Bitwise Operators	Description	Example
& (Bitwise AND)	Compares two bits and returns 1 if both of them are 1 or else returns 0	00111000 Returns 00011000
~ (Bitwise NOT)	Inverts every bits of the operand and is a unary operator	~00010101 Returns 11101010
| (Bitwise OR)	Compares two bits and returns 1 if the corresponding bits of either or both the operands is 1	00111000 Returns 00111100

- The Code Snippet demonstrates the use of bitwise operators.

```
//(56 = 00111000 and 28 = 00011100)
alert("56" + ' & ' + "28" + ' = ' + (56 & 28));
//(56 = 00111000 and 28 = 00011100)
alert("56" + ' | ' + "28" + ' = ' + (56 | 28));
```

Special Operators 1-2

There are some operators in JavaScript which do not belong to any of the categories of JavaScript operators.

Such operators are referred to as the special operators.

- Following table lists the special operators in JavaScript.

Special Operators	Description
, (comma)	Combines multiple expressions into a single expression, operates on them in the left to right order and returns the value of the expression on the right.
?: (conditional)	Operates on three operands where the result depends on a condition. It is also called as ternary operator and has the form condition, ? value1:value2. If the condition is true, the operator obtains value1 or else obtains value2.
typeof	Returns a string that indicates the type of the operand. The operand can be a string, variable, keyword, or an object.

Special Operators 2-2

- The Code Snippet demonstrates the use of special operator.

```
<SCRIPT>
  var age = parseInt(prompt(“Enter age”, “Age”))
  status = ((typeof(age) == “number” && (age >= 18))
  ?“eligible” : “not eligible”;
  document.write(‘You are ‘ + age + ‘ years old, so you are ‘
+status + ‘ to vote.’);
</SCRIPT>
```

Operator Precedence

- Following table lists the precedence of the operators from the highest to the lowest and their associativity.

Precedence Order	Operator	Description	Associativity
1	()	Parentheses	Left to Right
2	++, --	Post-increment and Post-decrement operators	Not Applicable
3	typeof, ++, --, -, ~, !	Pre-increment and Pre-decrement operators, Logical NOT, Bitwise NOT, and Unary negation	Right to Left
4	*, /, -	Multiplication, Division, and Modulo	Left to Right
5	+, -	Addition and Subtraction	Left to Right
6	<, <=, >, >=	Less than, Less than or equal, Greater than, and Greater than or equal	Left to Right
7	==, ===, !=, !==	Equal to, Strict equal to, Not equal to, and Strict not equal to	Left to Right
8	&, |, ^, &&, ||	Bitwise AND, Bitwise OR, Bitwise XOR, Logical AND, and Logical OR	Left to Right
9	?:	Conditional operator	Right to Left
10	=, +=, -=, *=, /=, %=	Assignment operators	Right to Left
11	,	Comma	Left to Right

Regular Expressions

Is a pattern that is composed of set of strings, which is to be matched to a particular textual content.

Allow handling of textual data effectively as it allows searching and replacing strings.

Allows handling of complex manipulation and validation, which could otherwise be implemented through lengthy scripts.

- There are two ways to create regular expressions which are as follows:

Literal Syntax:

- Refers to a static value
- Allows specifying a fixed pattern, which is stored in a variable
- Syntax is as follows:
  - var variable_name = /regular_expression_pattern/;

RegExp() Constructor:

- Is useful when the Web page designer does not know the pattern at the time of scripting.
- Method dynamically constructs a regular expression when the script is executed.
- Syntax is as follows:
- var variable_name = new RegExp(“regular_expression_pattern”,“flag”);

RegExp Methods and Properties

- RegExp object supports methods that are used for searching the pattern in a string. They are as follows:

test(string) - Tests a string for matching a pattern and returns a Boolean value of true or false. The Boolean value indicates whether the pattern exists or not in the string. The method is commonly used for validation.

exec(string) - Executes a string to search the matching pattern within it. The method returns a null value, if pattern is not found. In case of multiple matches, it returns the matched result set.

RegExp Methods and Properties

- RegExp object also supports properties that are used to get information regarding the string.
- Following table lists the properties of the RegExp object:

Audio Attributes	Description
$n	Represents the number from 1 to 9. It stores the recently handled parts of a parenthesized pattern of a regular expression.
aif	Indicates whether the given regular expression contain a g flag. The g flag specifies that all the occurrences of a pattern will be searched globally, instead of just searching for the first occurrence.
aifc	Indicates whether the given regular expression contains an i flag**.**
aiff	Stores the location of the starting character of the last match found in the string. In case of no match, the value of the property is -1.
asc	Stores the copy of the pattern.

RegExp Methods and Properties

- The Code Snippet displays the use of a regular expression.

```
<SCRIPT>
  var zipcodepattern = /^\d{5}$/;
  var zipcode = zipcodepattern.exec(prompt('Enter ZIP Code:'));
  if(zipcode != null)
  {
    alert('Valid ZIP Code.');
    alert('Regular Expression Pattern: ' + zipcodepattern.source);
   }
  else
  {
    alert('Invalid ZIP Code – Format xxxxx.');
  }
</SCRIPT>
```

Categories of Pattern Matching

- Different categories of pattern matching character that are required to create a regular expression pattern are as follows:
  - Position Matching
  - Character Classes
  - Repetition
  - Alternation and Grouping
  - Back Reference

Position Matching

Characters or symbols in this category allow matching a substring that exists at a specific position within a string.

- Following table lists the various position matching symbols.

Symbol	Description	Example
^	Denotes the start of a string	/^Good/ matches “Good” in “Good night”, but not in “A Good Eyesight”
$	Denotes the end of a string	/art$/ matches “art” in “Cart” but not in “artist”
\b	Matches a word boundary. A word boundary includes the position between a word and the space	/ry\b/ matches “ry” in “She is very good”
\B	Matches a non-word boundary	/\Ban/ matches “an” in “operand” but not in “anomaly”

Character Classes 1-2

Characters or symbols in this category are combined to form character classes for specifying patterns.

These classes are formed by placing a set of characters within the square brackets.

- Following table lists the various character classes symbols.

Symbol	Description	Example
[xyz]	Matches one of the characters specified within the character set	/^Good/ matches “Good” in “Good night”, but not in “A Good Eyesight”
[^xyz]	Matches one of the characters not specified within the character set	/[^BC]RT/ Matches “RRT” but, not “BRT” or “CRT”
.	Denotes a character except for the new line and line terminator	/s.t/ Matches “sat”, “sit”, “set”, and so on
\w	Matches alphabets and digits along with the underscore	/\w/ Matches “600” in “600%”

Character Classes 2-2

- Following table lists some more character classes symbols.

Symbol	Description	Example
\W	Matches a non-word character	/\W/ Matches “%” in “800%”
\d	Matches a digit between 0 to 9	/\d/ Matches “4” in “A4”
\D	Searches for a non-digit	/\D/ Matches “ID” in “ID 2246”
\s	Searches any single space character including space, tab, form feed, and line feed	/\s\w*/ Matches “ bar” in “scroll bar”
\S	Searches a non-space character	/\S\w*/ Matches “scroll” in “scroll bar”

Repetition

Characters or symbols in this category allow matching characters that reappear frequently in a string.

- Following table lists the various repetition matching symbols.

Symbol	Description	Example
{x}	Matches x number of occurrences of a regular expression	/\d{6}/ Matches exactly 6 digits”
{x, }	Matches either x or additional number of occurrences of a regular expression	/\s{4,}/ Matches minimum 4 whitespace characters
{x,y}	Matches minimum x to maximum y occurrences of a regular expression	/\d{6,8}/ Matches minimum 6 to maximum 8 digits
?	Matches minimum zero to maximum one occurrences of a regular expression	/l\s?m/ Matches “lm” or “l m”
*	Matches minimum zero to multiple occurrences of a regular expression	/im*/ Matches “i” in “Ice” and “imm” in “immaculate”, but nothing in “good”

Alternation and Grouping

Characters or symbols in this category allow grouping characters as an individual entity or adding the 'OR' logic for pattern matching.

- Following table lists the various alternation and grouping character symbols.

Symbol	Description	Example
()	Organizes characters together in a group to specify a set of characters in a string	/(xyz)+(uvw)/ Matches one or more number of occurrences of "xyz" followed by one occurrence of "uvw"
|	Combines sets of characters into a single regular expression and then matches any of the character set	/(xy)|(uv)|(st)/ Matches "xy" or "uv" or "st"

Back References

Characters or symbols in this category allow grouping characters as an individual entity or adding the 'OR' logic for pattern matching.

- Following table lists the various alternation and grouping character symbols.

Symbol	Description	Example
()\n	Matches a parenthesized set within the pattern, where n is the number of the parenthesized set to the left	/(\w+)\s+\1/ Matches any word occurring twice in a line, such as "hello hello". The \1 specifies that the word following the space should match the string, which already matched the pattern in the parentheses to the left of the pattern. To refer to more than one set of parentheses in the pattern, you would use \2 or \3 to match the appropriate parenthesized clauses to the left. You can have maximum 9 back references in the pattern

Decision-making Statements

Statements are referred to as a logical collection of variables, operators, and keywords that perform a specific action to fulfill a required task.

Statements help you build a logical flow of the script.

In JavaScript, a statement ends with a semicolon.

JavaScript is written with multiple statements, wherein the related statements are grouped together are referred to as block of code and are enclosed in curly braces.

Decision-making statements allow implementing logical decisions for executing different blocks to obtain the desired output.

They execute a block of statements depending upon a Boolean condition that returns either true or false.

Decision-making Statements

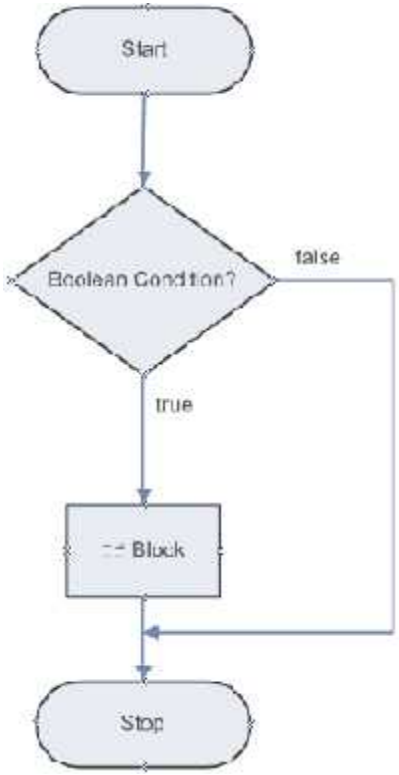
- JavaScript supports four decision-making statements, which are as follows:
  - `if`
  - `if-else`
  - `if-else if`
  - `switch`

if Statement 1-2

Executes a block of statements based on a logical Boolean condition.

If this condition is true, the block following the if statement is executed.

If the condition is false, the block after the if statement is not executed and the immediate statement after the block is executed.



if Statement 2-2

- The Code Snippet demonstrates the use of `if` statement.

```
<SCRIPT>
  var quantity = prompt('Enter quantity of product:',0);
  if(quantity < 0 || isNaN(quantity))
  {
    alert('Please enter a positive number.');
  }
</SCRIPT>
```

if-else Statement 1-2

if statement specifies a block of statement to be executed when the condition in the if statement is true.

Sometimes it is required to define a block of statements to be executed when a condition is evaluated to false.

if-else statement begins with the if block, which is followed by the else block.

The else block begins with the else keyword followed by a block of statements to be executed upon the false condition.

if-else Statement 2-2

- The Code Snippet demonstrates the use of `if-else` statement.

```
<SCRIPT>
  var firstNumber = prompt('Enter first number:',0);
  var secondNumber = prompt('Enter second number',0);
  var result = 0;
  if (secondNumber == 0)
  {
    alert('ERROR Message: Cannot divide by zero.');
  }
  else
  {
    result = firstNumber/secondNumber;
    alert("Result: " + result);
  }
</SCRIPT>
```

if-else-if Statement 1-2

- Allows you to check multiple conditions and specify a different block to be executed for each condition.
- Flow of these statements begins with the if statement followed by multiple else if statements and finally by an optional else block.
- Entry point of execution in these statements begins with the if statement.
- If the condition in the if statement is false, the condition in the immediate else if statement is evaluated.
- Also referred to as the if-else-if ladder.

if-else-if Statement 2-2

- The Code Snippet demonstrates the use of `if-else-if` statement.

```
<SCRIPT>
  var percentage = prompt('Enter percentage:',0);
  if (percentage >= 60)
  {
    alert ('You have obtained the A grade.');
  }
  else if (percentage >= 35 && percentage < 60)
  {
    alert ('You have obtained the B class.');
  }
  else
  {
    alert ('You have failed');
  }
</SCRIPT>
```

Nested if Statement 1-2

- Comprises of multiple if statements within an if statement.
- Flow of the nested-if statements starts with the if statement, which is referred to as the outer if statement.
- Outer if statement consists of multiple if statements, which are referred to as the inner if statements.
- Inner if statements are executed only if the condition in the outer if statement is true.
- Each of the inner if statements is executed but, only if the condition in its previous inner if statement is true.

Nested if Statement 2-2

- The Code Snippet demonstrates the use of nested `if` statement.

```
<SCRIPT>
  var username = prompt('Enter Username:');
  var password = prompt('Enter Password:');
  if (username != "" && password != "")
  {
    if (username == "admin" && password == "admin123")
    {
      alert('Login Successful');
    }
    else
    {
     alert ('Login Failed');
    }
  }
</SCRIPT>
```

Switch-case Statement 1-2

A program becomes quite difficult to understand when there are multiple if statements.

To simplify coding and to avoid using multiple if statements, switch-case statement can be used.

switch-case statement allows comparing a variable or expression with multiple values.

Switch-case Statement 2-2

- The Code Snippet demonstrates the use of `switch-case` statement.

```
<SCRIPT>
  var designation = prompt('Enter designation:');
  switch (designation)
  {
    case 'Manager':
      alert ('Salary: $21000');
      break;
    case 'Developer':
      alert ('Salary: $16000');
      break;
    default:
      alert ('Enter proper designation.');
      break;
  }
</SCRIPT>
```

Summary

- An operator specifies the type of operation to be performed on the values of variables and expressions.
- JavaScript operators are classified into six categories based on the type of action they perform on operands.
- There are six category of operators namely, Arithmetic, Relational, Logical, Assignment, Bitwise, and Special operators.
- Operators in JavaScript have certain priority levels based on which their execution sequence is determined.
- A regular expression is a pattern that is composed of set of strings, which is to be matched to a particular textual content.
- In JavaScript, there are two ways to create regular expressions namely, literal syntax and RegExp() constructor.
- Decision-making statements allow implementing logical decisions for executing different blocks to obtain the desired output.

Q & A