Session: 13

Operators and Statements

Objectives

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

Basics of Operators 1-2

- An operation is an action performed on one or more values stored in variables, 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.
  - **Unary operators** - Operates on a single operand.
  - **Binary operators** - Operates on two operands.
  - **Ternary operators** - Operates on three operands.
- Value or variable on which the operation is performed is called an operand.

Basics of Operators 2-2

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

Arithmetic / Increment & Decrement Operators

- **Arithmetic Operators**:
  - are binary operators:
  - + (addition) – (subtraction) * (multiplication) / (divsiion) % (modulo)
  - allow to perform computations on numeric and string values on two operands.
- **Increment – Decrement Operators:**
  - are unary operators.
  - can be placed either before (pre-increment or pre-decrement) or after the operand (post-increment or post-decrement.).

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

Relational & Logical Operators

- **Relational Operators** are binary operators that make a comparison between two operands.
- They are == != === !== > >= < <=
- After making a comparison, they return a boolean value: true or false.
- Expression consisting of a relational operator is called as the relational expression or conditional expression.
- **Logical operators** are operators that perform logical operations: && (and), || (or) , ! (not)
- They belong to the category of relational operators, as they return a boolean value.

Assignment Operators

- **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.

Expressions	Description	Result
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

- Represent their operands in bits (0s and 1s) and perform operations on them.
- They return standard decimal values.

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

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

Special Operators

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

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)	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, 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.

Operator Precedence

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 searching and replacing strings effectively.
- Allows handling of complex manipulation and validation
- There are two ways to create regular expressions:
  - **Literal Syntax:**
    - Refers to a static value, allows specifying a fixed pattern, stored in a variable
    - Syntax :

      var **variable_name** = **/regular_expression_pattern/**;
  - **RegExp() Constructor:**
    - Is useful when designer does not know the pattern at the time of scripting.
    - Syntax :

      var **variable_name** = **new RegExp("regular_expression_pattern","flag");**

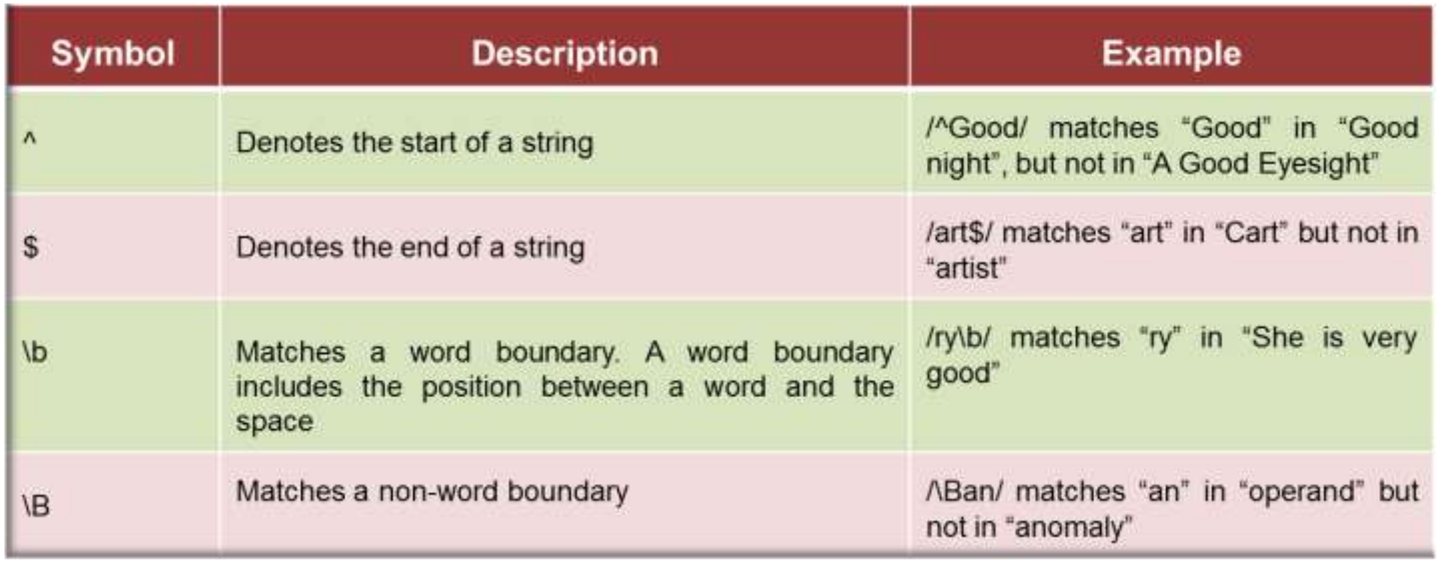
RegExp Methods and Properties

- **test(string)** - matches a pattern, returns a boolean value that indicates whether the pattern exists or not in the string, used for validation.
- **exec(string)** - searchs the matching pattern, returns a null value if pattern is not found. In case of multiple matches, returns the matched result set.

Attributes	Description
$n	Represents the number from 1 to 9. It stores the recently handled parts of a parenthesized pattern of a regular expression.
global	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.
IgnoreCase	Indicates whether the given regular expression contains an i flag.
lastndex	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.

Categories of Pattern Matching

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



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

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%"

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.

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

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

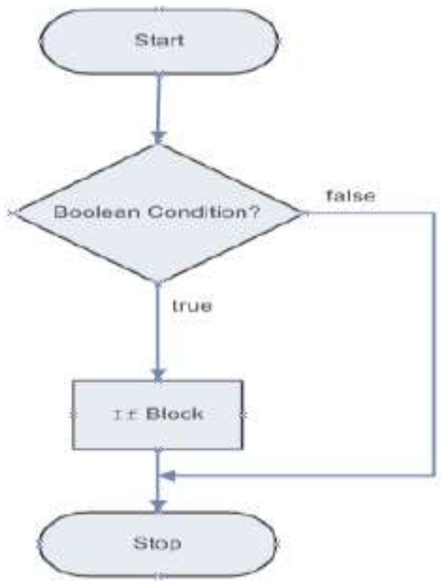
- Statements are referred to as a logical collection of variables, operators, and keywords that perform a specific action to fulfill a required task.
- Statements help you build a logical flow of the script.
- In JavaScript, a statement ends with a semicolon.
- The related statements are grouped together 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.

Decision-making Statements

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

- 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

```
<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

```
<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

```
<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

```
<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.