LESSON 3: JAVASCRIPT OPERATOR, FUNCTION AND OBJECTS

BY: DR. PAULINE ABESAMIS

SEPTEMBER 17, 2024

JAVASCRIPT DATA TYPES

- **JavaScript has 8 Datatypes**
- String
  Number
  Bigint
  Boolean
  Undefined
  Null
  Symbol
  Object

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

JAVASCRIPT OPERATORS

- Operators are used for JavaScript's arithmetic expressions, comparison expressions, logical expressions, assignment expressions, and more.
- Most operators are represented by punctuation characters such as + and =. Some, however are represented by keywords such as **delete** and **instanceof (known as "Key-word operators" or "regular operators")**
- Operators can be categorized based on the number of operands they expect (their "arity").
- Most JavaScript operators, like the * multiplication operator, are binary operators that combine two expressions into a single, more complex expression.
- Unary operator (single expression), supports ternary operator (conditional operator), combines three expressions into single expression.

JAVASCRIPT OPERATORS

Operator	Operation	A	N	Types
++	Pre- or post-increment	R	1	lval→num
--	Pre- or post-decrement	R	1	lval→num
-	Negate number	R	1	num→num
+	Convert to number	R	1	any→num
~	Invert bits	R	1	int→int
!	Invert boolean value	R	1	bool→bool
delete	Remove a property	R	1	lval→bool
typeof	Determine type of operand	R	1	any→str
void	Return undefined value	R	1	any→undef
**	Exponentiate	R	2	num,num→num
*, /, %	Multiply, divide, remainder	L	2	num,num→num
+, -	Add, subtract	L	2	num,num→num
+	Concatenate strings	L	2	str,str→str
<<	Shift left	L	2	int,int→int
>>	Shift right with sign extension	L	2	int,int→int
>>>	Shift right with zero extension	L	2	int,int→int
<, <=, >, >=	Compare in numeric order	L	2	num,num→bool
<, <=, >, >=	Compare in alphabetical order	L	2	str,str→bool
instanceof	Test object class	L	2	obj,func→bool
in	Test whether property exists	L	2	any,obj→bool
==	Test for non-strict equality	L	2	any,any→bool
!=	Test for non-strict inequality	L	2	any,any→bool
===	Test for strict equality	L	2	any,any→bool

Operator	Operation	A	N	Types
!==	Test for strict inequality	L	2	any,any→bool
&	Compute bitwise AND	L	2	int,int→int
^	Compute bitwise XOR	L	2	int,int→int
|	Compute bitwise OR	L	2	int,int→int
&&	Compute logical AND	L	2	any,any→any
||	Compute logical OR	L	2	any,any→any
??	Choose 1st defined operand	L	2	any,any→any
?:	Choose 2nd or 3rd operand	R	3	bool,any,any→any
=	Assign to a variable or property	R	2	lval,any→any
**=, *=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=, >>>=	Operate and assign	R	2	lval,any→any
,	Discard 1st operand, return 2nd	L	2	any,any→any

OPERATOR ASSOCIATIVITY

- "A specifies the associativity of the operator"
- A value of "L" specifies "left-to-right" associativity and value off "R" specifies the "right-to-left"
- "Precedence" are performed.

```
w = x - y - z;
```

is the same as:

```
w = ((x - y) - z);
```

```
y = a ** b ** c;
x = ~-y;
w = x = y = z;
q = a?b:c?d:e?f:g;
```

are equivalent to:

```
y = (a ** (b ** c));
x = ~(-y);
w = (x = (y = z));
q = a?b:(c?d:(e?f:g));
```

They are the same because the exponentiation, unary, assignment, and ternary conditional operators have right-to-left associativity

- JavaScript always evaluates expressions in strictly left-to-right order.
- Order of evaluation only makes a difference if any of the expressions being evaluated has side effects that affect the value of another expression

THE + OPERATOR

- The binary + operator adds numeric operands or concatenates string operands:

```
1 + 2                    // => 3
"hello" + " " + "there"  // => "hello there"
"1" + "2"                // => "12"
```

- When the values of both operands are numbers, or are both strings, then it is obvious what the + operator does
- Concatenation – if either the operands is a string or an object that converts to a string, the other operand is converted to a string and concatenation is performed.

```
1 + 2          // => 3: addition
"1" + "2"      // => "12": concatenation
"1" + 2        // => "12": concatenation after number-to-string
1 + {}         // => "1[object Object]": concatenation after object-to-string
true + true    // => 2: addition after boolean-to-number
2 + null       // => 2: addition after null converts to 0
2 + undefined  // => NaN: addition after undefined converts to NaN
```

when the + operator is used with strings and numbers, it may not be associative. That is, the result may depend on the order in which operations are performed.

```
1 + 2 + " blind mice"    // => "3 blind mice"
1 + (2 + " blind mice")  // => "12 blind mice"
```

UNARY ARITHMETIC OPERATORS

- Unary operators modify the value of a single operand to produce a new value.
- Unary operators all have high precedence and are all right-associative. The arithmetic unary operators described in this section (+, -, ++, and --) all convert their single operand to a number, if necessary.
- The unary arithmetic operators are:
  - Unary plus (+) - The unary plus operator converts its operand to a number (or to NaN) and returns that converted value.
  - Unary minus (-) it converts its operand to a number, if necessary, and then changes the sign of the result.
  - Increment (++) The ++ operator increments (i.e., adds 1 to) its single operand, which must be an lvalue (a variable, an element of an array, or a property of an object)

```
let i = 1, j = ++i;     // i and j are both 2
let n = 1, m = n++;     // n is 2, m is 1
```

- Decrement (--) It expects an lvalue operand. It converts the value of the operand to a number, subtracts 1, and assigns the decremented value back to the operand

RELATIONAL EXPRESSIONS

- These operators test for a relationship (such as "equals," "less than," or "property of ") between two values and return true or false depending on whether that relationship exists.
- Relational expression always evaluate to a boolean value, and that value is often used to control the flow of program execution in if, while, and for statements.

Equality and Inequality Operators

The =, ==, and === operators

JavaScript supports =, ==, and === operators. Be sure you understand the differences between these assignment, equality, and strict equality operators, and be careful to use the correct one when coding! Although it is tempting to read all three operators as "equals," it may help to reduce confusion if you read "gets" or "is assigned" for =, "is equal to" for ==, and "is strictly equal to" for ===.

The == operator is a legacy feature of JavaScript and is widely considered to be a source of bugs. You should almost always use === instead of ==, and !== instead of !=.

COMPARISON OPERATORS

Less than (<)

The `<` operator evaluates to `true` if its first operand is less than its second operand; otherwise, it evaluates to `false`.

Greater than (>)

The `>` operator evaluates to `true` if its first operand is greater than its second operand; otherwise, it evaluates to `false`.

Less than or equal (<=)

The `<=` operator evaluates to `true` if its first operand is less than or equal to its second operand; otherwise, it evaluates to `false`.

Greater than or equal (>=)

The `>=` operator evaluates to `true` if its first operand is greater than or equal to its second operand; otherwise, it evaluates to `false`.

The operands of these comparison operators may be of any type. Comparison can be performed only on numbers and strings, however, so operands that are not numbers or strings are converted.

```
1 + 2        // => 3: addition.
"1" + "2"    // => "12": concatenation.
"1" + 2      // => "12": 2 is converted to "2".
11 < 3       // => false: numeric comparison.
"11" < "3"   // => true: string comparison.
"11" < 3     // => false: numeric comparison, "11" converted to 11.
"one" < 3    // => false: numeric comparison, "one" converted to NaN.
```

THE IN OPERATOR

- The in operator expects a left-side operand that is a string, symbol, or value that can be converted to a string. It expects a right-side operand that is an object.
- It evaluates to true if the left-side value is the name of a property of the right-side object.
- For example:

```
let point = {x: 1, y: 1};  // Define an object
"x" in point               // => true: object has property named "x"
"z" in point               // => false: object has no "z" property.
"toString" in point        // => true: object inherits toString method

let data = [7,8,9];        // An array with elements (indices) 0, 1, and 2
"0" in data                // => true: array has an element "0"
1 in data                  // => true: numbers are converted to strings
3 in data                  // => false: no element 3
```

THE INSTANCEOF OPERATOR

- The instanceof operator expects a left-side operand that is an object and a right-side operand that identifies a class of objects.
- The operator evaluates to true if the left side object is an instance of the right-side class and evaluates to false otherwise.
- The classes of objects are defined by the constructor function that initializes them. Below are the examples:

```
let d = new Date();  // Create a new object with the Date() constructor
d instanceof Date    // => true: d was created with Date()
d instanceof Object  // => true: all objects are instances of Object
d instanceof Number  // => false: d is not a Number object
let a = [1, 2, 3];   // Create an array with array literal syntax
a instanceof Array   // => true: a is an array
a instanceof Object  // => true: all arrays are objects
a instanceof RegExp  // => false: arrays are not regular expressions
```

ASSIGNMENT WITH OPERATION

- Besides the normal = assignment operator, JavaScript supports a number of other assignment operators that provide shortcuts by combining assignment with some other operation.

```
total += salesTax;
```

is equivalent to this one:

```
total = total + salesTax;
```

Operator	Example	Equivalent
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Operator	Example	Equivalent
**=	a **= b	a = a ** b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
>>>=	a >>>= b	a = a >>> b
&=	a &= b	a = a & b
|=	a |= b	a = a | b
^=	a ^= b	a = a ^ b

JAVASCRIPT FUNCTIONS

- Functions are a fundamental building block for JavaScript programs and a common feature in almost all programming language.
- A *function* is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times.
- JavaScript functions are parameterized: a function definition may include a list of identifiers, known as parameters.
- Functions provide values, or arguments for the function's parameters. Functions often use their argument values to compute a return value that becomes the value of the function-invocation expression.
- Functions designed to initialize a newly created object are called "constructors".
- Functions are objects, and they can be manipulated by programs (e.g. priorities and invoke)

FUNCTION DECLARATION

- An identifier that names the function. The name is a required part of function declarations: it is used as the name of a variable, and the newly defined function object is assigned to the variable.
- A pair of parentheses around a comma-separated list of zero or more identifiers. These identifiers are the parameter names for the function, and they behave like local variables within the body of the function.
- A pair of curly braces with zero or more JavaScript statements inside. These statements are the body of the function: they are executed whenever the function is invoked.

JAVASCRIPT FUNCTION SYNTAX /BENEFIT

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
// Function returns the product of a and b
  return a * b;
}
```

- With functions you can reuse code
- You can write code that can be used many times.
- You can use the same code with different arguments, to produce different results.

EXAMPLE FUNCTION DECLARATION

```
// Print the name and value of each property of o.  Return undefined.
function printprops(o) {
    for(let p in o) {
        console.log(`${p}: ${o[p]}\n`);
    }
}
// Compute the distance between Cartesian points (x1,y1) and (x2,y2).
function distance(x1, y1, x2, y2) {
    let dx = x2 - x1;
    let dy = y2 - y1;
    return Math.sqrt(dx*dx + dy*dy);
}

// A recursive function (one that calls itself) that computes factorials
// Recall that x! is the product of x and all positive integers less than it.
function factorial(x) {
    if (x <= 1) return 1;
    return x * factorial(x-1);
}
```

Note: Function declarations is that the name of the function becomes a variable whose value is the function itself.

FUNCTION EXPRESSIONS

The code inside the function will execute when "something" **invokes** (calls) the function:

1. When an event occurs (when a user clicks a button)
2. When it is invoked (called) from JavaScript code
3. Automatically (self invoked)

```
// This function expression defines a function that squares its argument.
// Note that we assign it to a variable
const square = function(x) { return x*x; };

// Function expressions can include names, which is useful for recursion.
const f = function fact(x) { if (x <= 1) return 1; else return x*fact(x-1); };

// Function expressions can also be used as arguments to other functions:
[3,2,1].sort(function(a,b) { return a-b; });

// Function expressions are sometimes defined and immediately invoked:
let tensquared = (function(x) {return x*x;}(10));
```

SAMPLE FUNCTION

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
// Function returns the product of a and b
  return a * b;
}
```

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}

let value = toCelsius(77);
```

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```