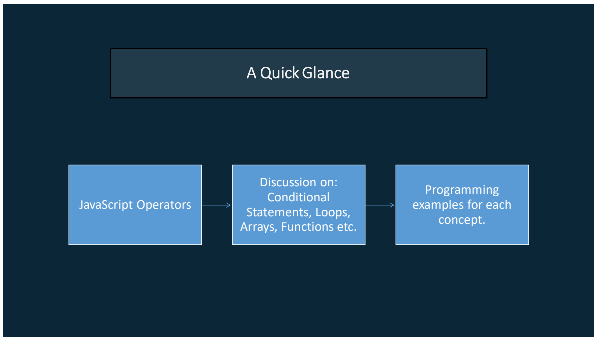
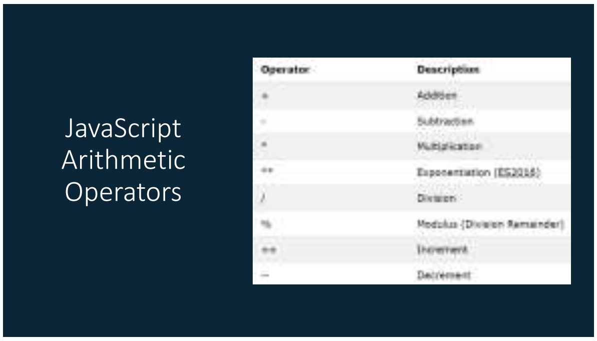
JavaScript: Part B

Learning with Examples



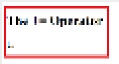
A Quick Glance

JavaScript Operators → Discussion on: Conditional Statements, Loops, Arrays, Functions etc. → Programming examples for each concept.



JavaScript Arithmetic Operators

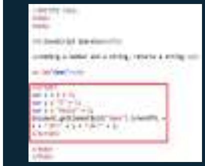
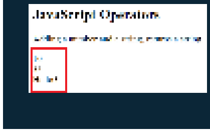
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement



JavaScript Assignment Operators

JavaScript String Operators

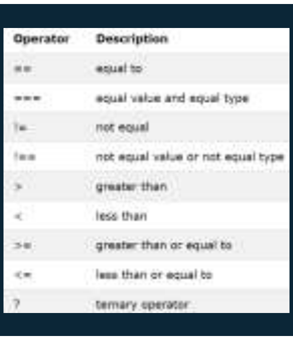
- The + operator can also be used to add (concatenate) strings.

Adding Strings and Numbers

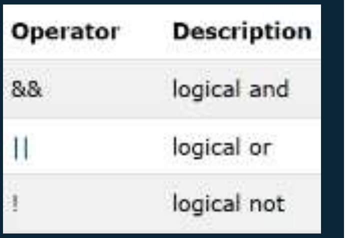
- Adding two numbers, will return a **sum**, but adding a number and a string will return a **string**

JavaScript Comparison Operators



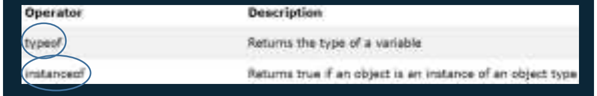
Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators



Operator	Description
&&	logical and
||	logical or
!	logical not

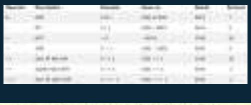
JavaScript Type Operators

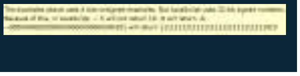


Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

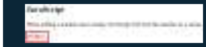
JavaScript Bitwise Operators

- Bit operators work on 32-bit number
- Any numeric operand in the operation is converted into a 32-bit number.
  - The result is converted back to a JavaScript number





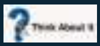
Electronics Tutorials



The Concept of Data Types

- In programming, data types are an important concept
- To be able to operate on variables, it is important to know something about the type.
- Without data types, a computer cannot safely solve this:

var x = 16 + "Volvo";



- Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?
- JavaScript will treat the example above as:

var x = "16" + "Volvo";

- When adding a number and a string, JavaScript will treat the number as a string.

JavaScript Types are Dynamic

- JavaScript has dynamic types. This means that the same variable can be used to hold different data types

var x; x is undefined
x = 5; x is a Number
x = "John"; x is a String

JavaScript Strings

- A string (or a text string) is a series of characters like "John Doe".
- Strings are written with quotes. You can use single or double quotes:

```
var carName1 = "Volvo XC60";   // Using double quotes
var carName2 = 'Volvo XC60';   // Using single quotes
```

- You can use quotes inside a string, as long as they don't match the quotes surrounding the string.

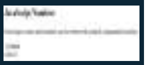
JavaScript Numbers

- JavaScript has only one type of number.
- Numbers can be written with, or without decimals:

```
var x1 = 34.00;     // Written with decimals
var x2 = 34;       // Written without decimals
```

- Extra large or extra small numbers can be written with scientific (exponential) notation:

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```



Activity: Implement both these programs using the online JavaScript editor to check the output and play with this program by replacing 5 by any other number.

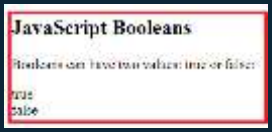
JavaScript Booleans

Booleans can have only two values: True and False

```
var x = 5;
var y = 5;
var z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Booleans are often used in conditional testing.

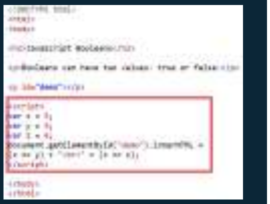
JavaScript Booleans Example



JavaScript Booleans

Booleans can have two values: true or false:

true
false

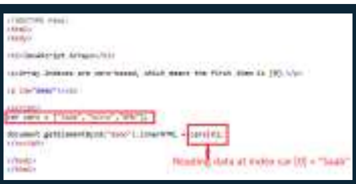


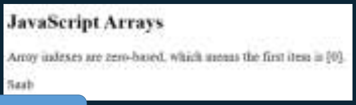
Now go to the online JavaScript editor and verify the output

JavaScript Arrays

- JavaScript arrays are written with square brackets
- Array items are separated by commas
- The following code declares (creates) an array called cars, containing three items (car names)
- var cars = ["Saab", "Volvo", "BMW"];
- Array indexes are zero-based, which means the first item is [0], second is [1], and so on

JavaScript Arrays Example





JavaScript Arrays

Array indexes are zero-based, which means the first item is [0].

Saab

Now go to the online JavaScript editor

JavaScript Objects Example

JavaScript objects are written with curly braces {}.

Object properties are written as name: value pairs, separated by commas.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

The object (person), in the example above, has 4 properties: firstName, lastName, age, and eyeColor.

Now go to the online JavaScript editor

JavaScript Objects

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- function myFunction(p1, p2) {
  return p1 * p2; // The function returns the product of p1 and p2
  }

Now go to the online JavaScript editor to verify the output

JavaScript Functions

JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***

The code to be executed, by the function, is placed inside curly brackets: **{}**

function *name* (*parameter1, parameter2, parameter3*) { *// code to be executed}*

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 Return

- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":
- Example

  Calculate the product of two numbers, and return the result:

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

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

22

Why Functions?

- You can reuse code: Define the code once and use it many times.
- You can use the same code many times with different arguments, to produce different results.
- Example

  Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

23

Functions Used as Variable Values

- Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.
- Example: Instead of using a variable to store the return value of a function:

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

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

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

JavaScript Functions

The temperature is 25 Celsius

Now go to the online JavaScript Editor

24

Local Variables

- Variables declared within a JavaScript function, become **LOCAL** to the function.
- Local variables can only be accessed from within the function.
- Example

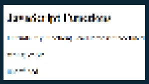
```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

- Since local variables are only recognised inside their functions, variables with the same name can be used in different functions.
- Local variables are created when a function starts, and deleted when the function is completed.

Now go to the online JavaScript Editor



JavaScript if else and else if

Conditional statements are used to perform different actions based on different conditions.

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

Use if to specify a block of code to be executed, if a specified condition is true.

Use else to specify a block of code to be executed, if the same condition is false

Use else if to specify a new condition to test, if the first condition is false

Use switch to specify many alternative blocks of code to be executed

The if Statement

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

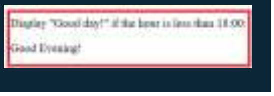
Syntax

- if (*condition*) {
  // *block of code to be executed if the condition is true*
  }

Example: Make a "Good day" greeting if the hour is less than 18:00:

```
if (hour < 18) {
  greeting = "Good day";
}
```

Now go to the online JavaScript Editor and verify the output



```
Display "Good day!" if the hour is less than 18:00

Good Evening!
```

The else Statement

- The else statement is used to specify a block of code to be executed if the condition is false.
- Syntax:

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

- Example: If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Now go to the online JavaScript Editor

The else if Statement

- Use the else if statement statement to specify a new condition if the first condition is false.
- Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is
true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is
false
}
```

- Example: If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Now go to the online JavaScript Editor

The JavaScript Switch Statement

The switch statement is used to perform different actions based on different conditions.

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

Example of switch statement

- The getDay () method returns the weekday as a number between 0 and 6.

  (Sunday=0, Monday=1, Tuesday=2 ..)
- This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

Now go to the online JavaScript Editor

JavaScript Loops

- Loops are handy, if you want to run the same code over and over again, each time with a different value.
- Often this is the case when working with arrays:
- Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

- You can write:

```
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

Now go to the online JavaScript Editor

Different Kinds of Loops

- JavaScript supports different kinds of loops:
  - for: loops through a block of code a number of times
  - for/in: loops through the properties of an object
  - for/of: loops through the values of an iterable object
  - while: loops through a block of code while a specified condition is true
  - do/while: also loops through a block of code while a specified condition is true

The For Loop

- The for loop has the following syntax:

  for (*statement 1*; *statement 2*; *statement 3*) {
  // code block to be executed
  }
- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed.
- Example

  for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
  }
- From the example above, you can read:
  - Statement 1 sets a variable before the loop starts (var i = 0).
  - Statement 2 defines the condition for the loop to run (i must be less than 5).
  - Statement 3 increases a value (i++) each time the code block in the loop has been executed.

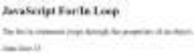
Now go to the online JavaScript Editor

The For/In Loop

- The JavaScript for/in statement loops through the properties of an object.
- Example

  var person = {fname:"John", lname:"Doe", age:25};
  var text = "";
  var x;
  for (x in person) {
  text += person[x];
  }

Now go to the online JavaScript Editor



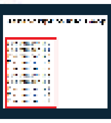
The For/Of Loop

The JavaScript for/of statement loops through the values of an iterable objects.

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The for/of loop has the following syntax:

for (*variable* of *iterable*) {
// code block to be executed
}



The While Loop

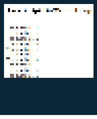
- The while loop loops through a block of code as long as a specified condition is true.
- Syntax

```
while (condition) {
  // code block to be executed
}
```

- In the following example, the code in the loop will run, over and over again, as long as variable (i) is less than 10:

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

Now go to the online JavaScript Editor



The Do/While Loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax

```
do {
  // code block to be executed
}
while (condition);
```

- The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Now go to the online JavaScript Editor

JavaScript Break and Continue

The break statement "jumps out" of a loop.

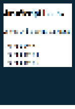
The continue statement "jumps over" one iteration in the loop.

The Break Statement

- The break statement can be used to jump out of a loop.
- The break statement breaks the loop and continues executing the code after the loop (if any):
- Example:

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

Now go to the online JavaScript Editor



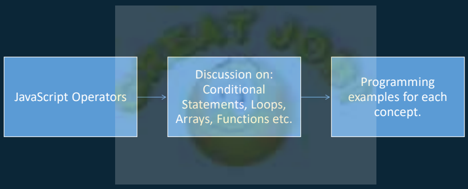
The Continue Statement

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- This example skips the value of 3:

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

Now go to the online JavaScript Editor

Overview



JavaScript Operators

Discussion on: Conditional Statements, Loops, Arrays, Functions etc.

Programming examples for each concept.