Web Engineering Lab

Lab 06

JavaScript: Basics, Functions, Loops & Decision Making

Instructor: Hurmat Hidayat
Course Code: SL3003
Semester Spring 2023

Department of Computer Science,

National University of Computer and Emerging

Sciences Peshawar Campus

Lab Content

JavaScript for Web Page

A Web developers usually talk about three languages that are used to create web pages: HTML, CSS, and JavaScript. Each language forms a separate layer with a different purpose.

- **Content Layer** : This is where the content of the page lives. The HTML gives the page structure and adds semantics.
- **Presentation Layer**: The CSS enhances the HTML page with rules that state how the HTML content is presented (backgrounds, borders, box dimensions, colors, fonts, etc.).
- **Behaviour Layer**: This is where we can change how the page behaves, adding interactivity. We aim to keep as much of our JavaScript as possible in separate files.

Create and Link JavaScript file

A JavaScript file is just a text file (like HTML and CSS files are) but it has a . js file extension. When you want to use JavaScript with a web page, you use the HTML <script> element to tell the browser it is coming across a script. Its src attribute tells people where the JavaScript file is stored. You may see JavaScript in the HTML between opening <script> and closing </script> tags (but it is better to put scripts in their own files).

Example:

add-content.js

```
var today = new Date();
var hourNow = today.getHours();
var greeting;

if (hourNow > 18) {
    greeting = 'Good evening!';
} else if (hourNow > 12) {
    greeting = 'Good afternoon!';
} else if (hourNow > 0) {
    greeting = 'Good morning!';
} else {
    greeting = 'Welcome!';
}

document.write('<h3>' + greeting + '</h3>');
```

Add-content.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp;amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp;amp; Co.</h1>
    <script src="js/add-content.js"></script>
    <p>For all orders and inquiries please call <em>555-3344</em></p>
  </body>
</html>
```

JavaScript runs where it is found in the HTML. When the browser comes across a <script> element, it stops to load the script and then checks to see if it needs to do anything. If you view the source code of the page in the browser, the JavaScript will not have changed the HTML, because the script works with the model of the web page that the browser has created.

Basic JavaScript

Statements

A script is a series of instructions that a computer can follow one-by-one. Each individual instruction or step is known as a statement.

- Statements should end with a semicolon.
- JavaScript is case sensitive so hourNow means something different to HourNow or HOURNOW. A statement is an individual instruction , each one should start on a new line and end with a semicolon. This makes your code easier to read and follow.
- Some statements are surrounded by curly braces; these are known as code blocks. The closing curly brace is not followed by a semicolon.

Comments

You should write comments to explain what your code does. They help make your code easier to read and understand. This can help you and others who read your code.

Multiline Comments: To write a comment that stretches over more than one line, you use a multi-line comment, starting with the /* characters and ending with the * / characters. Anything between these characters is not processed by the JavaScript interpreter.

Single Line Comments: In a single-line comment, anything that follows the two forward slash characters I/ on that line will not be processed by the JavaScript interpreter. Single line comments are often used for short descriptions of what the code is doing.

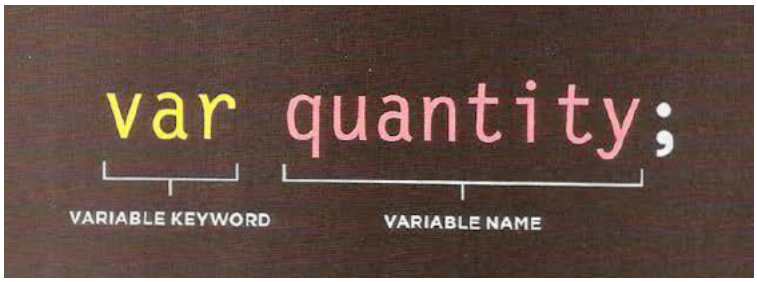
```
/* This script displays a greeting to the user based upon the current time.
It is an example from JavaScript & jQuery book */

var today= new Date();            // Create a new date object
var hour Now = today.getHours();  // Find the current hour
var greeting;

//Display the appropriate greeti ng based on the current time
if (hourNow > 18) {
    greeting= 'Good evening!';
else if (hourNow > 12) {
    greeting = ' Good afternoon!';
else if (hourNow > 0) {
    greeting = 'Good morni ng!';
else {
    greeting = 'Welcome! ' ;
}

document .write( '<h3>' +greeting + '</ h3>');
```

Variable

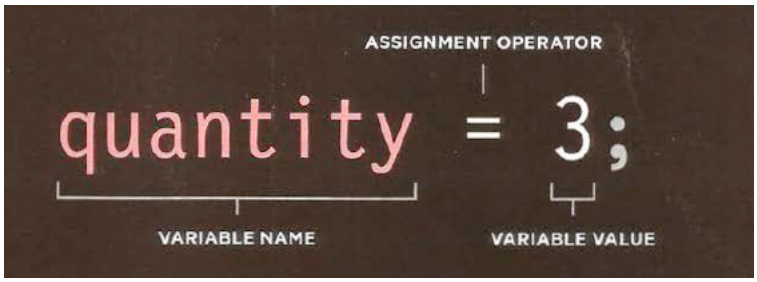
A script will have to temporarily store the bits of information it needs to do its job. It can store this data in variables.

Variable Declaration

Before you can use a variable, you need to announce that you want to use it. This involves creating a variable and giving it a name. We can declare variable as follow:



Assign Value to a Variable



Data Types

JavaScript distinguishes between numbers, strings, and true or false values known as Booleans.

NUMERIC DATA TYPE:

The numeric data type handles numbers. For example: 0.75 For tasks that involve counting or calculating sums, you will use numbers 0-9. For example, five thousand, two hundred and seventy-two would be written 5272 (note there is no comma between the thousands and the

hundreds). You can also have negative numbers (such as -23678) and decimals (three quarters is written as 0.75).

STRING DATA TYPE:

The strings data type consists of letters and other characters. For example: 'Hi, Alen!'

Note how the string data type is enclosed within a pair of quotes. These can be single or double quotes, but the opening quote must match the closing quote. Strings can be used when working with any kind of text. They are frequently used to add new content into a page and they can contain HTML markup.

```
var username;
var message;
username = 'Molly';
message = 'See our upcoming range';
```

Sometimes you will want to use a double or single quote mark within a string. Because strings can live in single or double quotes, if you just want to use double quotes in the string, you could surround the entire string in single quotes. If you just want to use single quotes in the string, you could surround the string in double quotes.

```
var title;
var message;

title= "Molly's Special Offers" ;
message = '<a href=\"sale .html\">25% off l</a>' ;

var elTitle = document.getElementByld('title') ;
elTitle.innerHTML =title;

var elNote = document.getElementByid ('note') ;
elNote.innerHTML =message;
```

BOOLEAN DATA TYPE:

Boolean data types can have one of two values: true or false. For example: true

Booleans are used when your code can take more than one path.

```
var inStock;
var shipping;
inStock = true;
shipping= false;
```

Note: Unlike some other programming languages, when declaring a variable in JavaScript, you do not need to specify what type of data it will hold.

Shorthand For Creating Variables

Programmers sometimes use shorthand to create variables. Here are three variations of how to declare variables and assign them values:

JAVASCRIPT c02/js/shorthand-variable.js

```
① var price = 5;
  var quantity = 14;
  var total = price * quantity;

② var price, quantity, total;
  price = 5;
  quantity = 14;
  total = price * quantity;

③ var price = 5, quantity = 14;
  var total = price * quantity;
```

Rules For Naming Variables

Here are six rules you must always follow when giving a variable a name:

- The name must begin with a letter, dollar sign ($),or an underscore (_). It must not start with a number.
- The name can contain letters, numbers, dollar sign ($), or an underscore (_). Note that you must not use a dash(-) or a period (.) in a variable name.
- You cannot use keywords or reserved words.
- All variables are case sensitive, so score and Score would be different variable names, but it is bad practice to create two variables that have the same name using different cases.
- Use a name that describes the kind of information that the variable stores. For example, firstName might be used to store a person's first name, lastNarne for their last name, and age for their age.
- If your variable name is made up of more than one word, use a capital letter for the first letter of every word after the first word.

Arrays

An array is a special type of variable. It doesn't just store one value; it stores a list of values. Arrays are especially helpful when you do not know how many items a list will contain because, when you create the array, you do not need to specify how many values it will hold. For example, an array can be suited to storing the individual items on a shopping list because it is a list of related items.

```
1  var colors;
2  colors ['white', 'black', ' custom '];
3
4  var el document.getElementByld('col ors');
5  el.textContent = col ors[0];
6
```

The values are assigned to the array inside a pair of square brackets, and each value is separated by a comma. The values in the array do not need to be the same data type, so you can store a string, a number and a Boolean all in the same array. This technique for creating an array is known as an **array literal**.

ACCESSING ITEMS IN AN ARRAY

To retrieve the third item on the list, the array name is specified along with the index number in square brackets.

```
1  var itemThree;
2  itemThree = colors [ 2] ;
3
```

Here you can see a variable called itemThree is declared. Its value is set to be the third color from the colors array.

NUMBER OF ITEMS IN AN ARRAY

Each array has a property called length, which holds the number of items in the array. The name of the array is followed by a period symbol (or full stop) which is then followed by the length keyword.

```
1  var numColors ;
2  numColors =colors.length;
3
```

Expression

An expression evaluates into (results in) a single value. Broadly speaking there are two types of expressions.

- Expression that just assign a value to a variable `var color = 'beige';`
- Expression that use two or more values to return a single value `var area = 3 * 2;`

Operators

Expressions rely on things called operators; they allow programmers to create a single value from one or more values. Operators are of the following types:

- Assignment Operators
- Arithmetic Operators
- String Operators
- Comparison Operators
- Logical Operators

Arithmetic Operators

JavaScript contains the following mathematical operators, which you can use with numbers.

NAME	OPERATOR	PURPOSE & NOTES	EXAMPLE	RESULT
ADDITION	+	Adds one value to another	10 + 5	15
SUBTRACTION	-	Subtracts one value from another	10 - 5	5
DIVISION	/	Divides two values	10 / 5	2
MULTIPLICATION	*	Multiplies two values using an asterisk (Note that this is not the letter x)	10 * 5	50
INCREMENT	++	Adds one to the current number	i = 10; i++;	11
DECREMENT	--	Subtracts one from the current number	i = 10; i--;	9
MODULUS	%	Divides two values and returns the remainder	10 % 3	1

Using Arithmetic Operators

```
// Create a variable for the subtotal and make a calculation
var subtotal = (13 + 1) * 5; // Subtotal is 70

// Create a variable for the shipping and make a calculation
var shipping = 0.5 * (13 + 1); // Shipping is 7

// Create the total by combining the subtotal and shipping values
var total = subtotal + shipping; // Total is 77

// Write the results to the screen
var elSub = document.getElementById('subtotal');
elSub.textContent = subtotal;

var elShip = document.getElementById('shipping');
elShip.textContent = shipping;

var elTotal = document.getElementById('total');
elTotal.textContent = total;

```

String Operator

There is just one string operator: the+ symbol. It is used to join the strings on either side of it. There are many occasions where you may need to join two or more strings to create a single value. Programmers call the process of joining together two or more strings to create one new string concatenation. For example

var firstName = 'Ivy' ;

var lastName = ' Stone' ;

var full Name = firstName + lastName;

Output: Ivy Stone

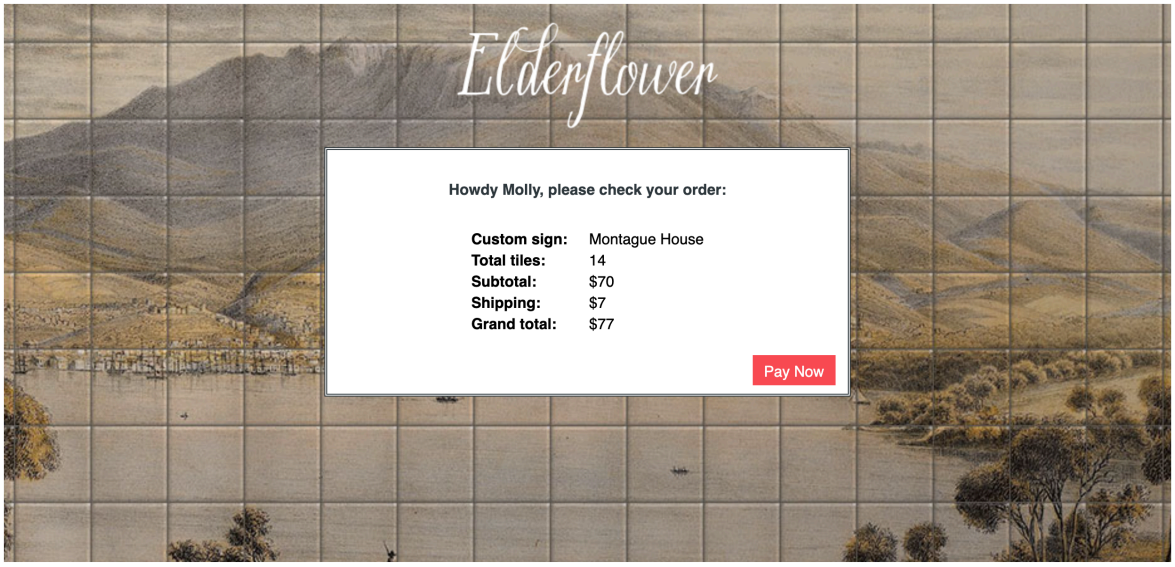
```
// Store the greeting in a variable
var greeting = 'Howdy ';

// Store the users name in a variable
var name = 'Molly';

/* Create the welcome message by concatenating the strings in the two variables *
var welcomeMessage = greeting + name + '!';

// Get the element that has an id of greeting
var el = document.getElementById('greeting');

// Replace the content of this element with the personal message
el.textContent = welcomeMessage;
```

Basic JavaScript Instruction Example

```
<html>
<head>
  <title> JavaScript Example</title>
  <link rel="stylesheet" href="css/c02.css" />
</head>
<body>
  <h1>Elderflower</h1>
  <div id="content">
    <div id="greeting" class="message">Hello!</div>
    <table>
      <tr>
        <td>Custom sign: </td>
        <td id="userSign"></td>
      </tr>
      <tr>
        <td>Total tiles: </td>
        <td id="tiles"></td>
      </tr>
      <tr>
        <td>Subtotal: </td>
        <td id="subTotal">$</td>
      </tr>
      <tr>
        <td>Shipping: </td>
        <td id="shipping">$</td>
      </tr>
      <tr>
        <td>Grand total: </td>
        <td id="grandTotal">$</td></tr>
    </table>
    <a href="#" class="action">Pay Now</a>
  </div>
  <script src="js/example.js"></script>
</body>
</html>
```

JavaScript File:

```
// Create variables for the welcome message
var greeting = 'Howdy ';
var name = 'Molly';
var message = ', please check your order:';
// Concatenate the three variables above to create the welcome message
var welcome = greeting + name + message;

// Create variables to hold details about the sign
var sign = 'Montague House';
var tiles = sign.length;
var subTotal = tiles * 5;
var shipping = 7;
var grandTotal = subTotal + shipping;

// Get the element that has an id of greeting
var el = document.getElementById('greeting');
// Replace the content of that element with the personalized welcome message
el.textContent = welcome;

// Get the element that has an id of userSign then update its contents
var elSign = document.getElementById('userSign');
elSign.textContent = sign;

// Get the element that has an id of tiles then update its contents
var elTiles = document.getElementById('tiles');
elTiles.textContent = tiles;

// Get the element that has an id of subTotal then update its contents
var elSubTotal = document.getElementById('subTotal');
elSubTotal.textContent = '$' + subTotal;

// Get the element that has an id of shipping then update its contents
var elShipping = document.getElementById('shipping');
elShipping.textContent = '$' + shipping;

// Get the element that has an id of grandTotal then update its contents
var elGrandTotal = document.getElementById('grandTotal');
elGrandTotal.textContent = '$' + grandTotal;

```



Functions or Methods

Functions let you group a series of statements together to perform a specific task. If different parts of a script repeat the same task, you can reuse the function (rather than repeating the same set of statements).

A Basic Function

In this example, the user is shown a message at the top of the page. The message is held in an HTML element whose id attribute has a value of message. The message is going to be changed using JavaScript.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Basic Function</title>
    <link rel="stylesheet" href="css/c03.css" />
  </head>
  <body>
    <h1>TravelWorthy</h1>
    <div id="message">Welcome to our site!</div>
    <script src="js/basic-function.js"></script>
  </body>
</html>
```

Before the closing </body> tag, you can see the link to the JavaScript file.

The JavaScript file starts with a variable used to hold a new message, and is followed by a function called updateMessage().

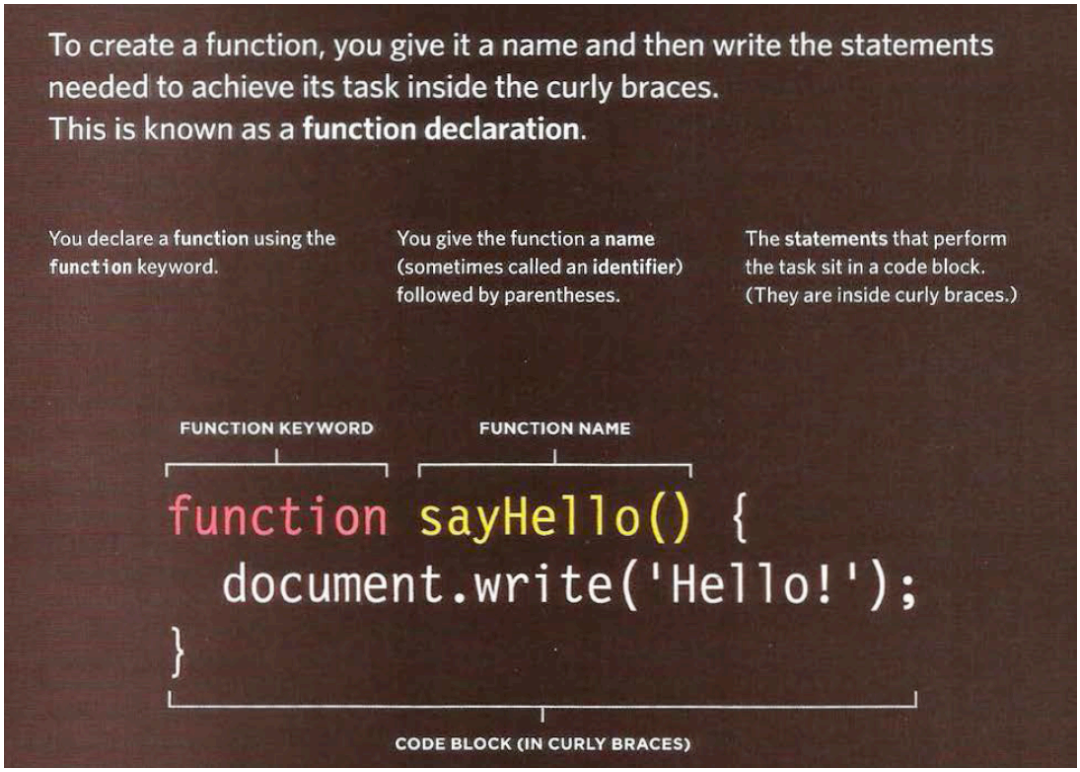
JavaScript

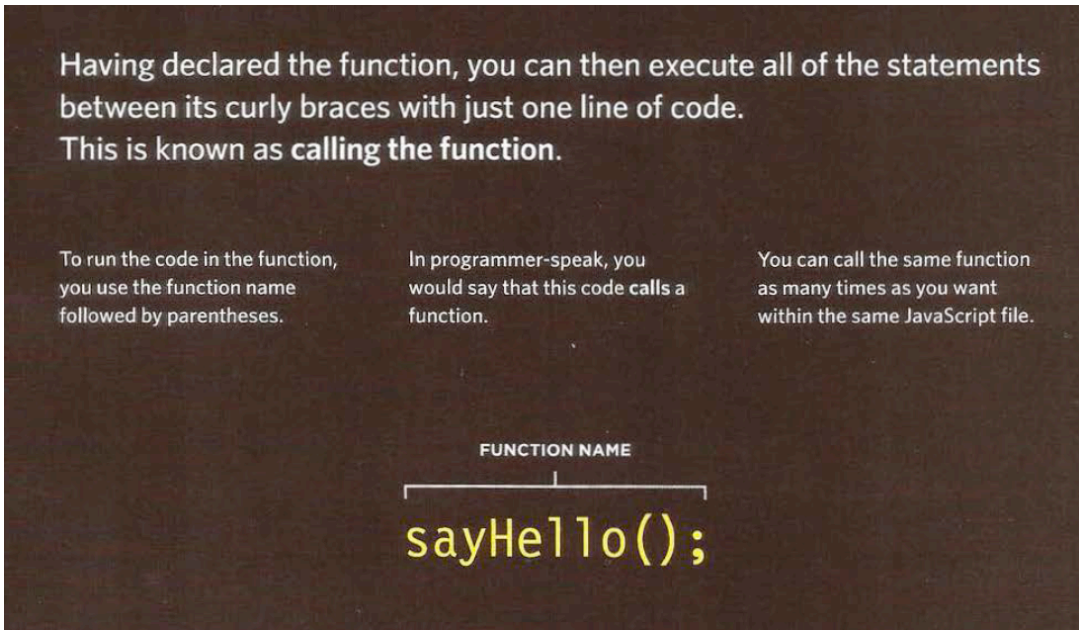
```
// Create a variable called msg to hold a new message
var msg = 'Sign up to receive our newsletter for 10% off!';

// Create a function to update the content of the element
// whose id attribute has a value of message

function updateMessage() {
  var el = document.getElementById('message');
  el.textContent = msg;
}

// Call the function
updateMessage();
```

Output

Declaring and Calling a Function

Function can store instructions for a specific task.

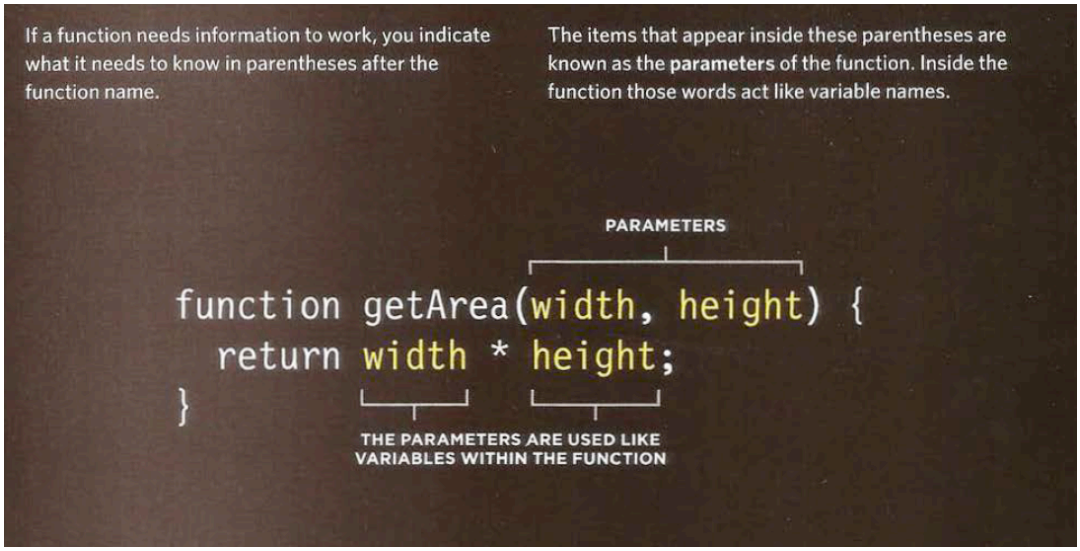




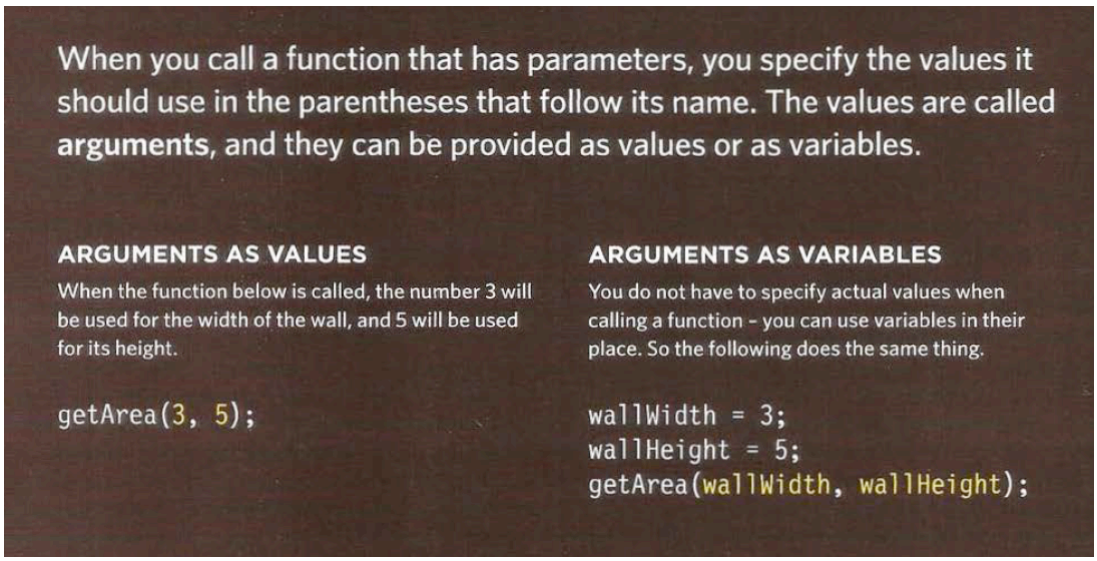
Sometimes you will see a function called before it has been declared . This still works because the interpreter runs through the script before executing each statement.

Parameters

Sometimes a function needs specific information to perform its task. In such cases, when you declare the function you give it parameters. Inside the function, the parameters act like variable.

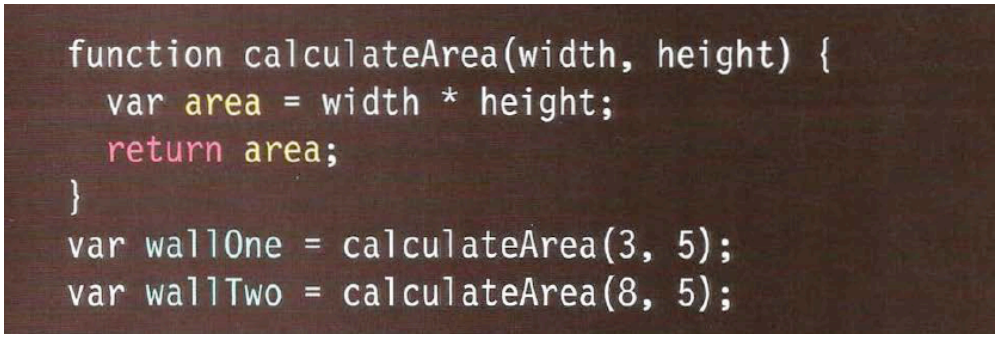


This function will calculate area of a rectangle. To do this, it needs the rectangle width and height. Each time you call this function values could be different.



Function Return

Some functions returns information to code that called them. For example when the perform calculation they return result. The calculateArea() function returns the area of rectangle to the code that called it.



Functions can return more than one value using an array. For example the function calculate area and volume of a box and returns the result.

```
function getSize(width, height, depth) {
  var area = width * height;
  var volume = width * height * depth;
  var sizes = [area, volume];
  return sizes;
}
var areaOne = getSize(3, 2, 3)[0];
var volumeOne = getSize(3, 2, 3)[1];
```

Anonymous Functions

If you put a function where the interpreter would expect to see an expression, then it is treated as an expression, and it is known as a function expression. In function expressions, the name is usually omitted. A function with no name is called an anonymous function. Below, the function is stored in a variable called area. It can be called like any function created with a function declaration.

```
var area = function(width, height) {
return width * height;
} ;
var size = area(3, 4) ;
```

Immediately Invoked Function Expression

Pronounced "iffy," these functions are not given a name. Instead, they are executed once as the interpreter comes across them. Below, the variable called area will hold the value returned from the function (rather than storing the function itself so that it can be called later).

```
var area = (function() {
  var width = 3;
  var height = 2;
  return width * height;
}());
```

The final parentheses (shown on green) after the closing curly brace of the code block tell the interpreter to call the function immediately. The grouping operators (shown on pink) are parentheses there to ensure the interpreter treats this as an expression.

Loops

Loops checks a condition. If it returns true, a code block will run. Then the condition will be checked again and if it still returns true, the code block will run again. It repeats until the condition returns false. There are three common types of loops:

1. for loop
2. while loop
3. Do while loop

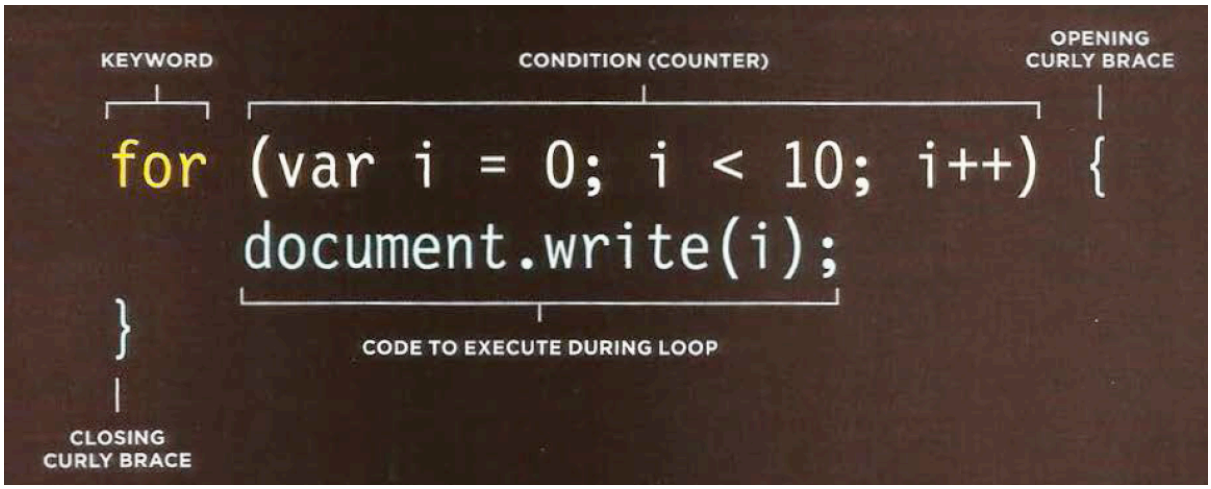
For Loop

If you need to run code a specific number of times, use for loop. In a for loop, the condition is usually a counter which is used to tell how many times the loop should run. Here you can see the condition is made up of three statements:

Initialization : var i = 0 ;

Condition : i<10;

Update: i++



Key Loop Concepts

You will commonly see these two keywords used with loops:

break:

This keyword causes the termination of the loop and tells the interpreter to go onto the next statement of code outside of the loop. (You may also see it used in functions.)

continue:

This keyword tells the interpreter to continue with the current iteration, and then check the condition again. (If it is true, the code runs again.)

Note: PERFORMANCE ISSUES

It is important to remember that when a browser comes across JavaScript, it will stop doing anything else until it has processed that script. If your loop is dealing with only a small number of items, this will not be an issue. If, however, your loop contains a lot of items, it can make the page slower to load.

Using For loop

In this example, the scores for each round of a test are stored in an array called scores. The total number of items in the array is stored in a variable called arrayl ength. This number is obtained using thelength property of the array. There are three more variables: roundNumber holds the round of the test; msg holds the message to display.

```
1  var scores = [24, 32, 17];      // Array of scores
2  var arrayLength = scores.length;// Items in array
3  var roundNumber = 0;            // Current round
4  var msg = '';                   // Message
5
6  // Loop through the items in the array
7  for (var i = 0; i < arrayLength; i++) {
8
9    // Arrays are zero based (so 0 is round 1)
10   // Add 1 to the current round
11   roundNumber = (i + 1);
12
13   // Write the current round to message
14   msg += 'Round ' + roundNumber + ': ';
15
16   // Get the score from the scores array
17   msg += scores[i] + '<br />';
18 }
19
20 document.getElementById('answer').innerHTML = msg;
21
```

Using While loop

Here is an example of a while loop. It writes out the 5 times table. Each time the loop is run, another calculation is written into the variable called msg. This loop will continue to run for as long as the condition in the parentheses is true. That condition is a counter indicating

that, as long as the variable i remains less than 10, the statements in the subsequent code block should run.

```
var i = 1;       // Set counter to 1
var msg = '';    // Message

// Store 5 times table in a variable
while (i < 10) {
  msg += i + ' x 5 = ' + (i * 5) + '<br />';
  i++;
}

document.getElementById('answer').innerHTML = msg;
```

Using do while loop

The key difference between a while loop and a do while loop is that the statements in the code block come before the condition. This means that those statements are run once whether or not the condition is met. If you take a look at the condition, it is checking that the value of the variable called i is less than 1, but that variable has already been set to a value of 1.

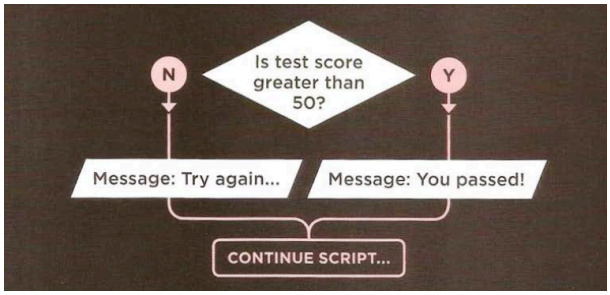
```
var i = 1;       // Set counter to 1
var msg = '';    // Message

// Store 5 times table in a variable
do {
  msg += i + ' x 5 = ' + (i * 5) + '<br />';
  i++;
} while (i < 1);
// Note how this is already 1 and it still runs

document.getElementById('answer').innerHTML = msg;
```

Decision Making

There are often several places in a script where decisions are made that determine which line of code should be run next.

Flow Chart

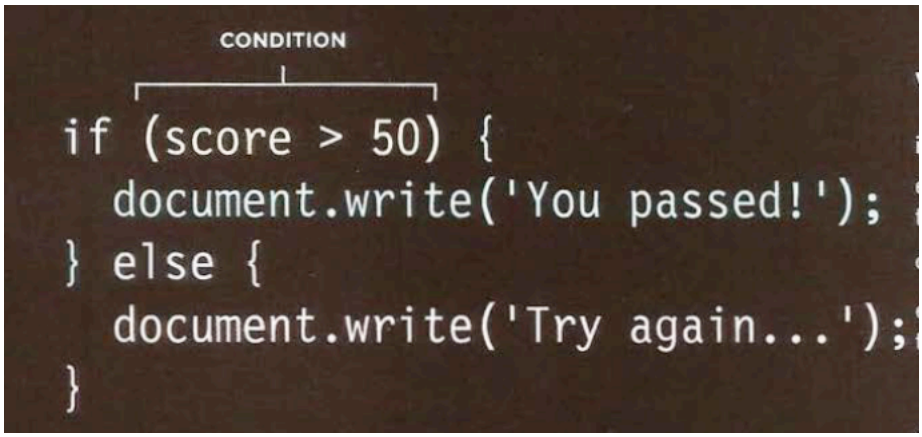
In a flowchart, the diamond shape represents a point where a decision must be made and the code can take one of the two different paths. In order to determine which path to take, you set a condition. If the condition returns true, you have to take one path; if it is false, you take another path. You have to write different set of codes for each situation/path.



Evaluating Conditions & Conditional Statements

There are two components to a decision:

1. A expression is evaluated, which returns a value
2. A conditional statement says what to do in a given situation



If the condition returns true, execute the statements between the first set of curly brackets otherwise Execute the statements between the second set of curly brackets.

Comparison Operators

The following comparison operators are used :

- Is equal to : ==

This operator compares two values (numbers, strings, or Boolean) to see if they are the same.

- Is not equal to : !=

This operator compares two values (numbers, strings, or Boolean) to see if they are not the same.

- Strict equal to: ===

This operator compares two values to check that both the data type and value are the same.

- Strict not equal to: !==

This operator compares two values to check that both the data type and value are not the same.

- Greater than: >

This operator checks if the number on the left is greater than the number on the right.

- Less than:

This operator checks if the number on the left is less than the number on the right.

- Greater than or equal to: >=

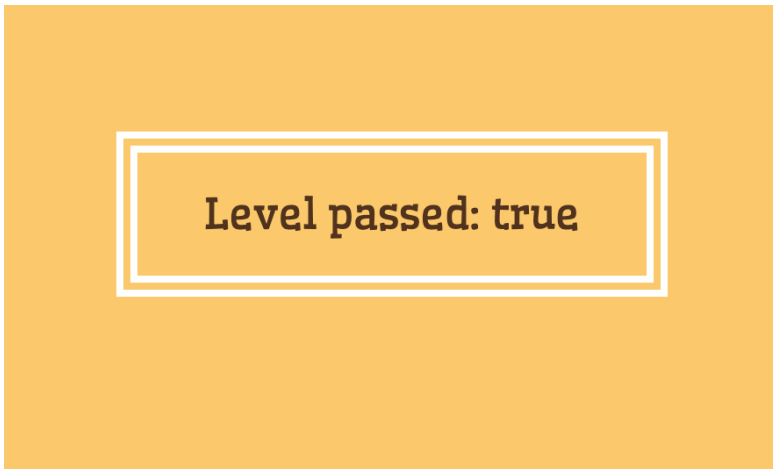
This operator checks if the number on the left is greater or equal than the number on the right.

- Less than or equal to:

This operator checks if the number on the left is less than or equal to the number on the right.

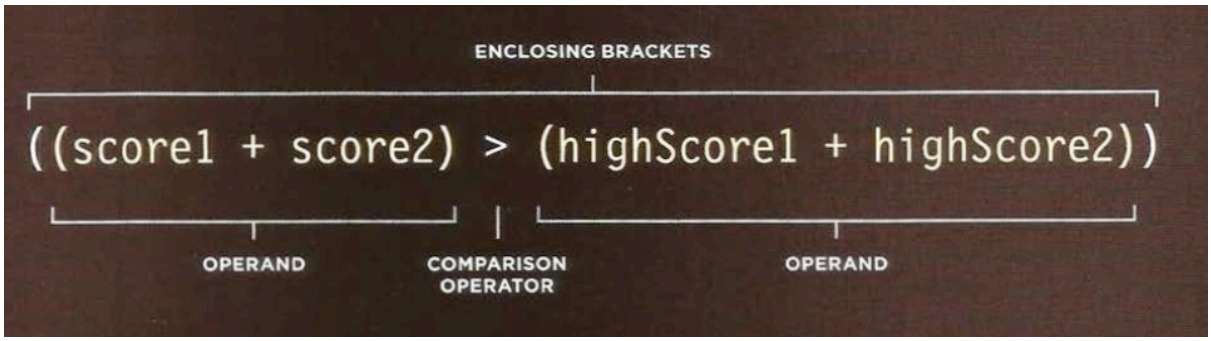
Using Comparison Operators

```
1   var pass = 50;   // Pass mark
2   var score = 90;  // Score
3
4   // Check if the user has passed
5   var hasPassed = score >= pass;
6
7   // Write the message into the page
8   var el = document.getElementById('answer');
9   el.innerHTML = 'Level passed: ' + hasPassed;
10
```



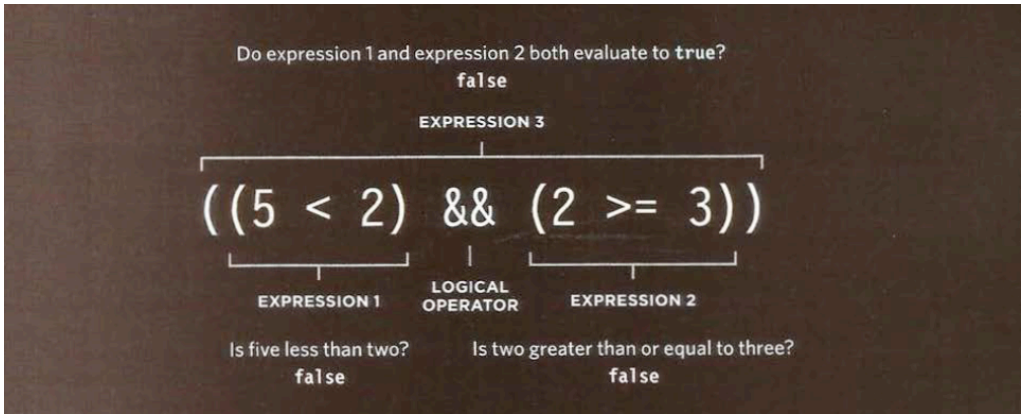
Using Expressions with Comparison Operators

The operands doesn't have to be a single value or variable name. An operand can be an expression.



Using Logical operators

Logical Operators allow you to compare the results of more than one comparison operator.



The following logical operators are used :

- Logical AND : &&

This operator tests more than one conditions. If all expressions returns true then the expression returns true.

- Logical OR : ||

This operator tests at least one condition. If either expression evaluates to true, then the expression returns true.

- Logical Not : !

This operator takes a single Boolean value and inverts it. This reverses the state of an expression.

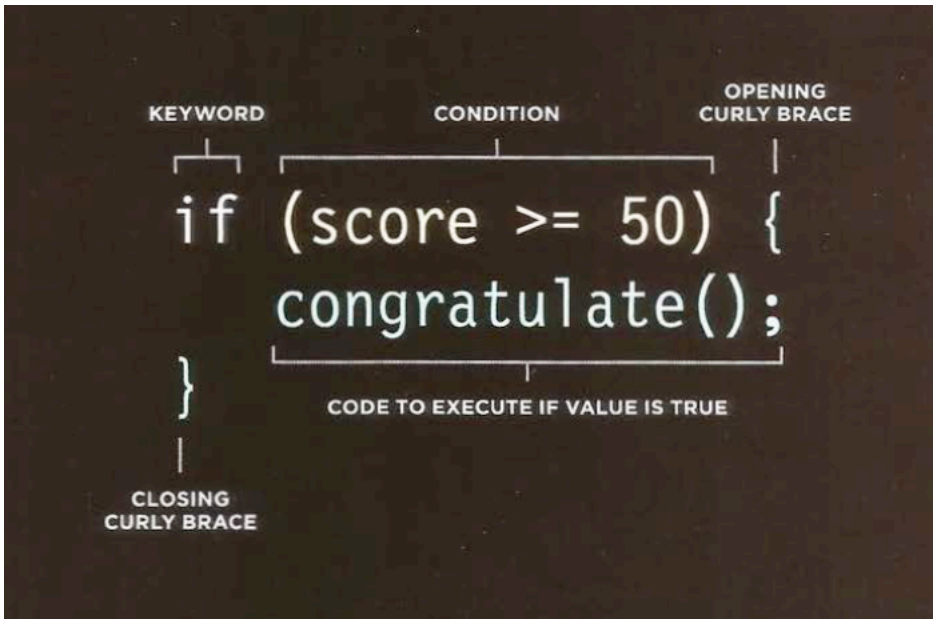
Example:

```
var score1 = 8;   // Round 1 score
var score2 = 8;   // Round 2 score
var pass1 = 6;    // Round 1 pass mark
var pass2 = 6;    // Round 2 pass mark

// Check whether user passed both rounds, store result in variable
var passBoth = (score1 >= pass1) && (score2 >= pass2);

// Create message
var msg = 'Both rounds passed: ' + passBoth;

// Write the message into the page
var el = document.getElementById('answer');
el.innerHTML = msg;
```

Both rounds passed:
true

IF Statements

The if statement evaluates (or checks) a condition. If the condition evaluates to true, any statements in the subsequent code block are executed.



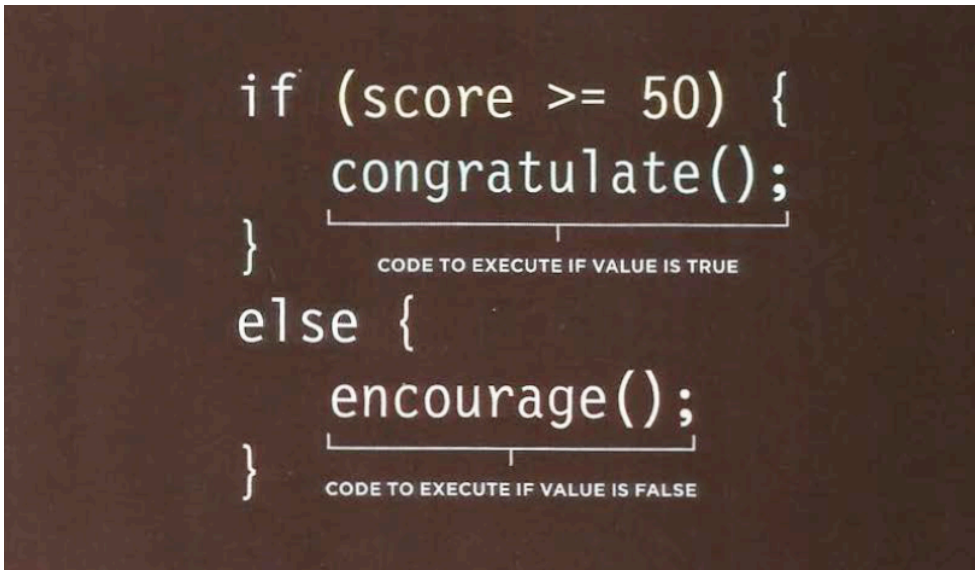
Using if Statements

```
1   var score = 75;      // Score
2   var msg;             // Message
3
4   if (score >= 50) {  // If score is 50 or higher
5     msg = 'Congratulations!';
6     msg += ' Proceed to the next round.';
7   }
8
9   var el = document.getElementById('answer');
10  el.textContent = msg;
11
```

Congratulations! Proceed to the next round.

IF .. ELSE Statements

The if ..else statement evaluates (or checks) a condition. If it resolves to true the first code block is executed. If the condition resolves to false the second code block is run instead.



Using if else Statements

```
1   var pass = 50;       // Pass mark
2   var score = 75;     // Current score
3   var msg;             // Message
4
5   // Select message to write based on score
6   if (score > pass) {
7     msg = 'Congratulations, you passed!';
8   } else {
9     msg = 'Have another go!';
10  }
11
12  var el = document.getElementById('answer');
13  el.textContent = msg;
14
```

Lab Tasks:

1. Create an HTML webpage, add the following two tables, then write a JavaScript program which compute, the average marks of the following students . Use this average to determine the corresponding grade. Display the average marks and grade on the web page.

Student Name	Marks
David	80
Vinoth	77
Divya	88
Ishitha	95
Thomas	68

Range	Grade
<60	F
<70	D
<80	C
<90	B
<100	A

2. Write a JavaScript function that takes two arrays of numbers as input and returns a new array that contains only the elements that are common to both arrays, in the order they appear in the first array. If there are no common elements, the function should return an empty array. For example:

   Input: **[1, 2, 3, 4]**, **[2, 4, 6, 8]** Output: **[2, 4]**

Input: **[1, 2, 3]**, **[4, 5, 6]** Output: **[]**