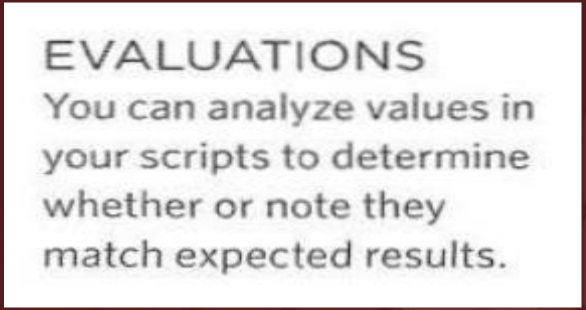
Frontend Development II

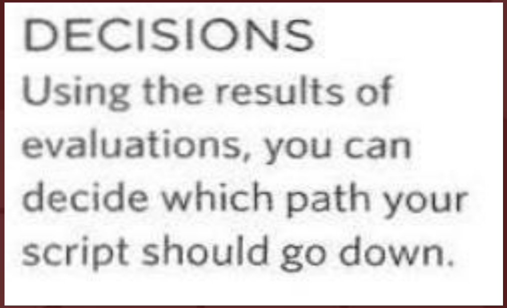
Decisions and Loops

Scripts often need to behave differently depending upon how the user interacts with the web page and/or the browser window itself. To determine which path to take, programmers often rely upon the following three concepts:



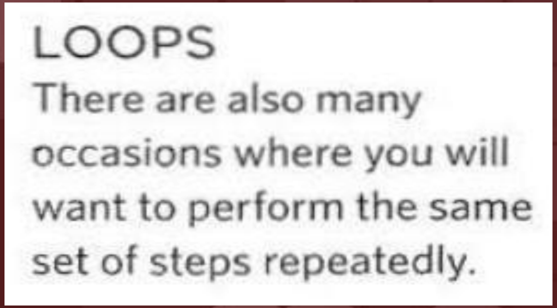
EVALUATIONS

You can analyze values in your scripts to determine whether or note they match expected results.



DECISIONS

Using the results of evaluations, you can decide which path your script should go down.



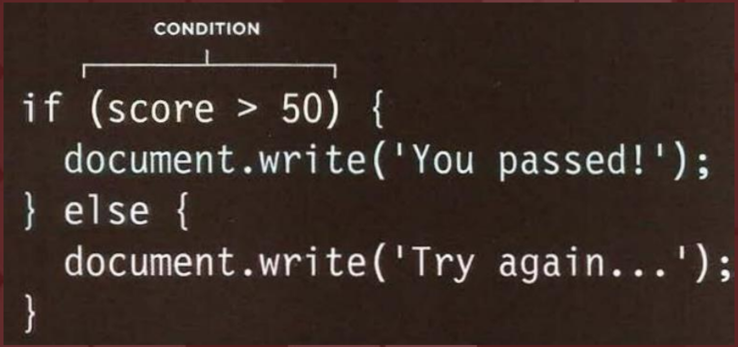
LOOPS

There are also many occasions where you will want to perform the same set of steps repeatedly.

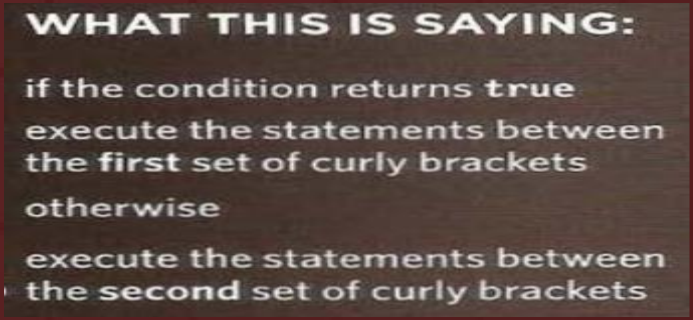
Evaluation conditions and Conditional Statements

There are two components to a decisions :

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



```
if (score > 50) {
  document.write('You passed!');
} else {
  document.write('Try again...');
}
```



WHAT THIS IS SAYING:

if the condition returns true

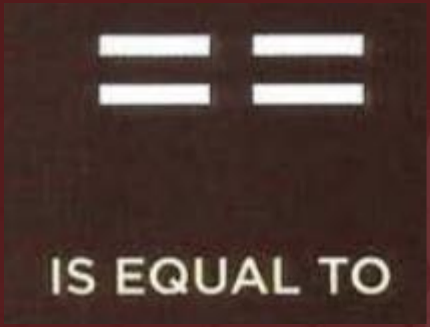
execute the statements between the first set of curly brackets

otherwise

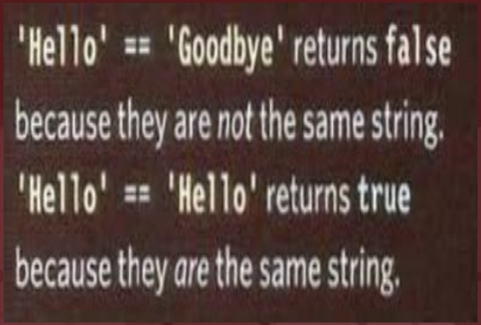
execute the statements between the second set of curly brackets

Comparison Operators :

You can evaluate a situation by comparing one value in the script to what you expect it might be. The result will be a Boolean : true or false


==

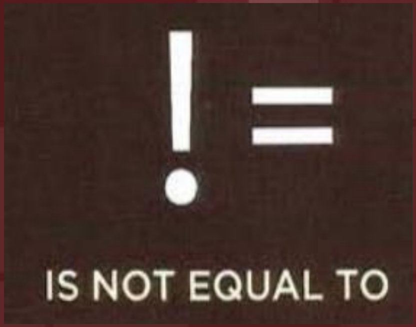
IS EQUAL TO


'Hello' == 'Goodbye' returns false

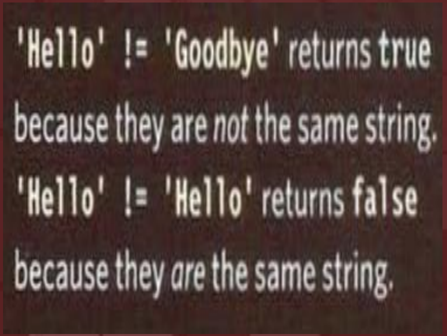
because they are *not* the same string.

'Hello' == 'Hello' returns true

because they *are* the same string.


!=

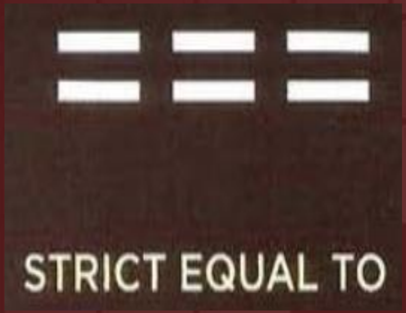
IS NOT EQUAL TO


'Hello' != 'Goodbye' returns true

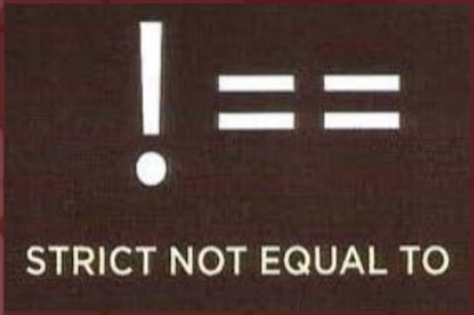
because they are *not* the same string.

'Hello' != 'Hello' returns false

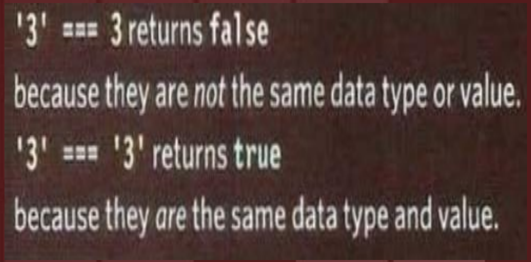
because they *are* the same string.


===

STRICT EQUAL TO


!==

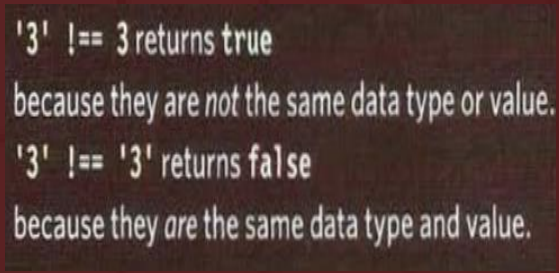
STRICT NOT EQUAL TO


'3' === 3 returns false

because they are *not* the same data type or value.

'3' === '3' returns true

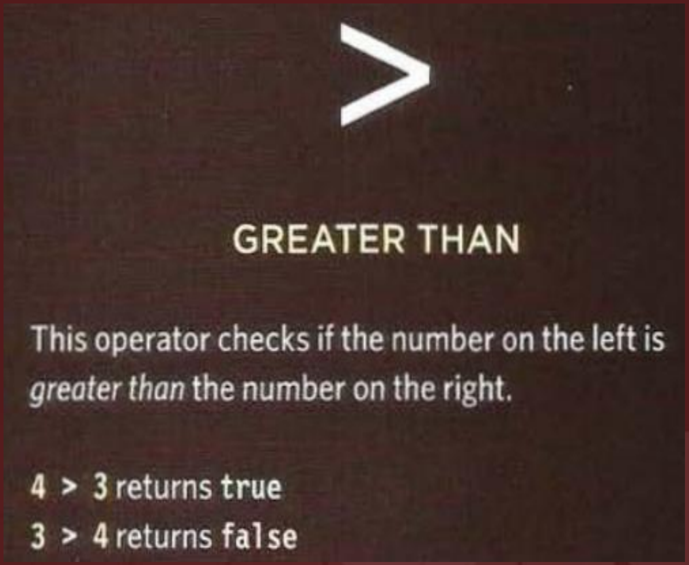
because they *are* the same data type and value.


'3' !== 3 returns true

because they are *not* the same data type or value.

'3' !== '3' returns false

because they *are* the same data type and value.

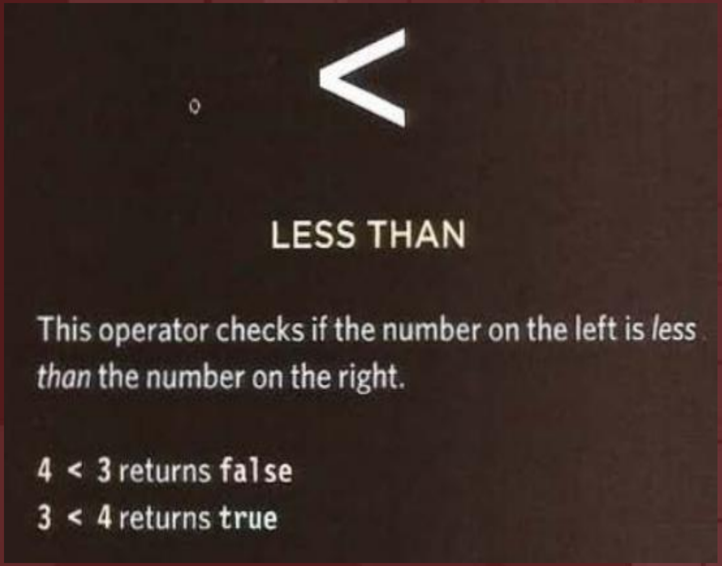


>

GREATER THAN

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

`4 > 3` returns `true`
`3 > 4` returns `false`

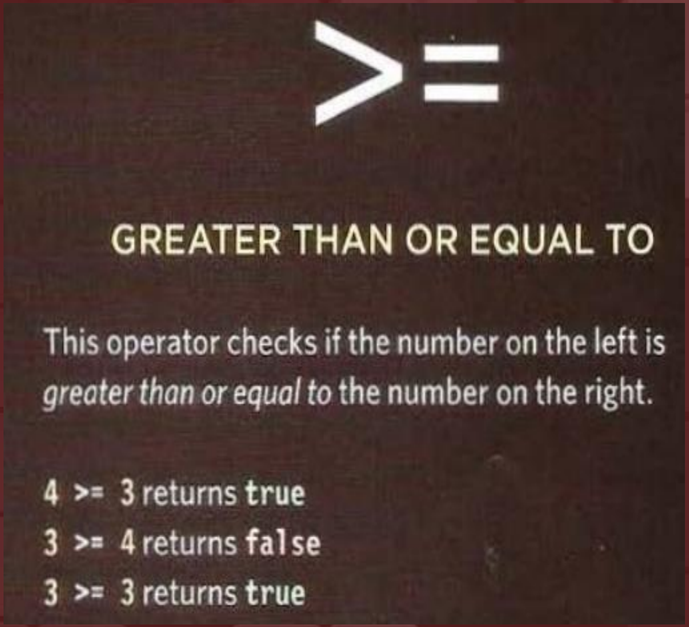


<

LESS THAN

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

`4 < 3` returns `false`
`3 < 4` returns `true`

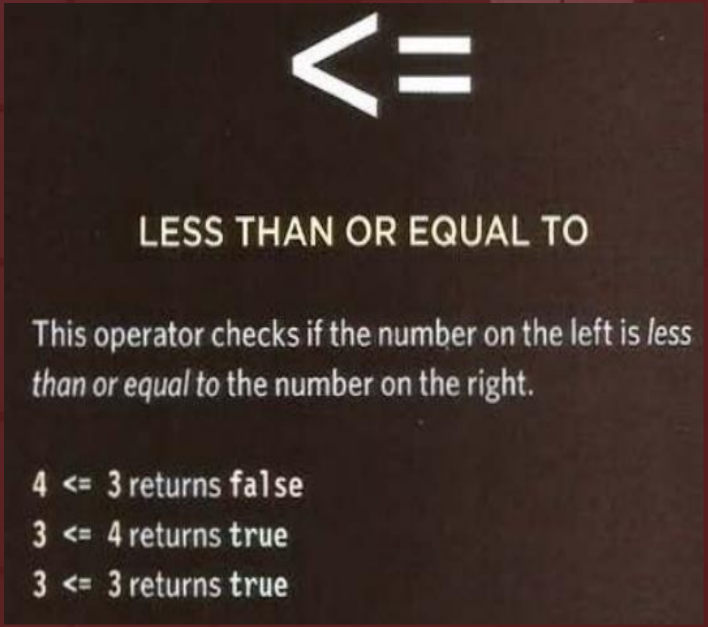


>=

GREATER THAN OR EQUAL TO

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

`4 >= 3` returns `true`
`3 >= 4` returns `false`
`3 >= 3` returns `true`

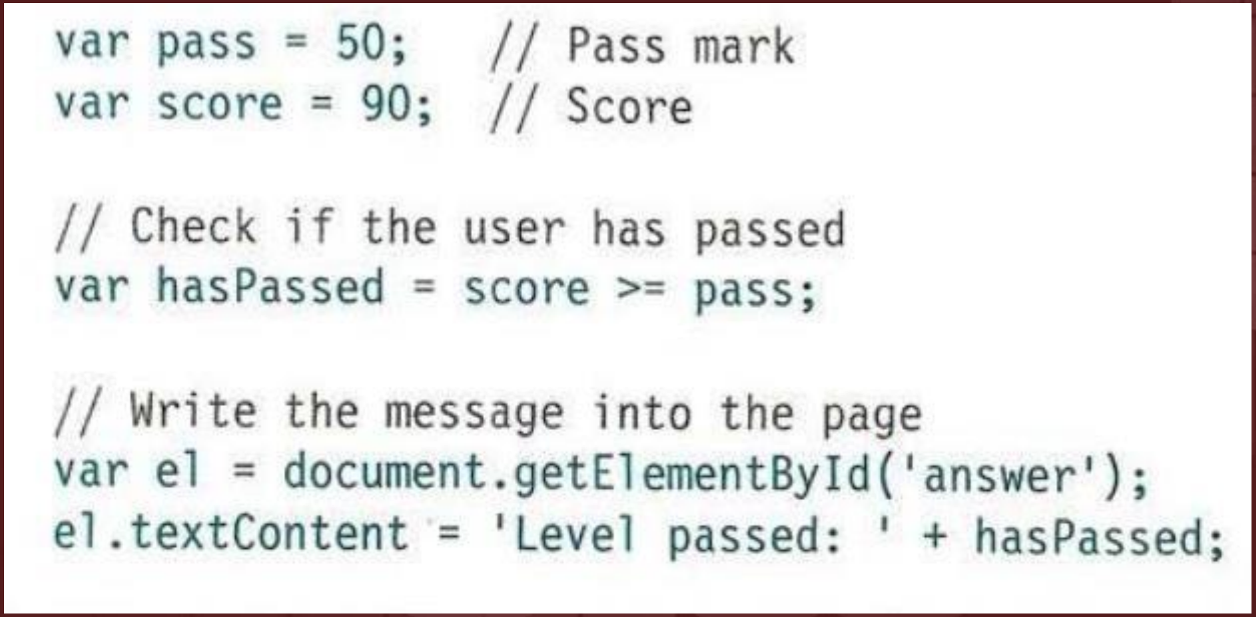


<=

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.

`4 <= 3` returns `false`
`3 <= 4` returns `true`
`3 <= 3` returns `true`

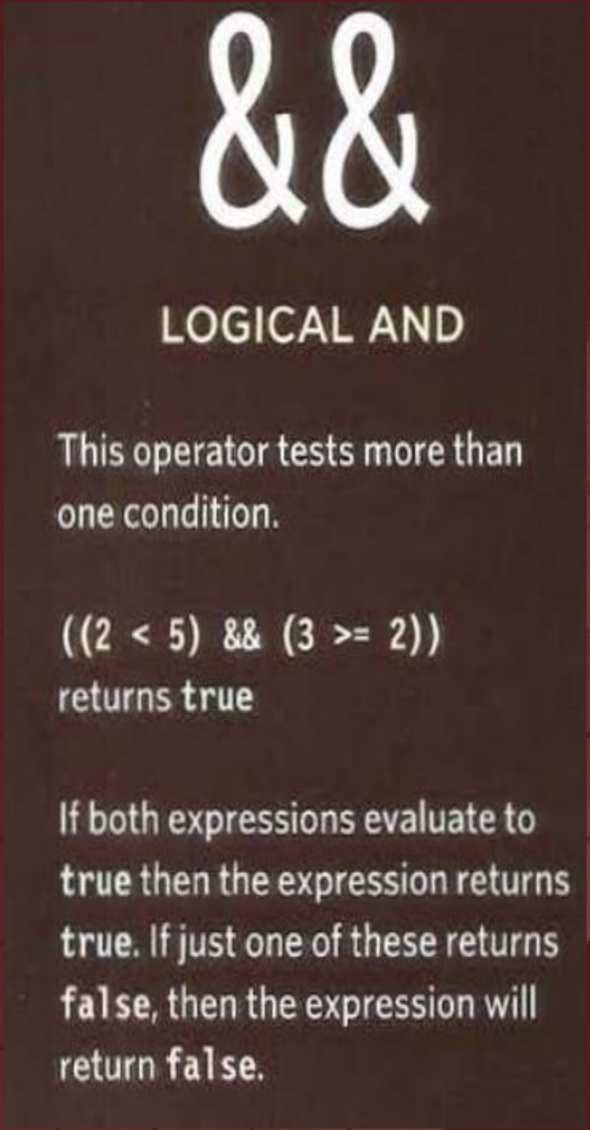


```
var pass = 50;    // Pass mark
var score = 90;   // Score

// Check if the user has passed
var hasPassed = score >= pass;

// Write the message into the page
var el = document.getElementById('answer');
el.textContent = 'Level passed: ' + hasPassed;
```

The example starts by setting
two variables:
1. pass to hold the pass mark
2. score to hold the users score

To see if the user has passed, a comparison operator checks whether score is greater than or equal to pass. The result will be true or false , and is stored in a variable called has Passed. On the next line, the result is written to the screen.

Logical Operators :



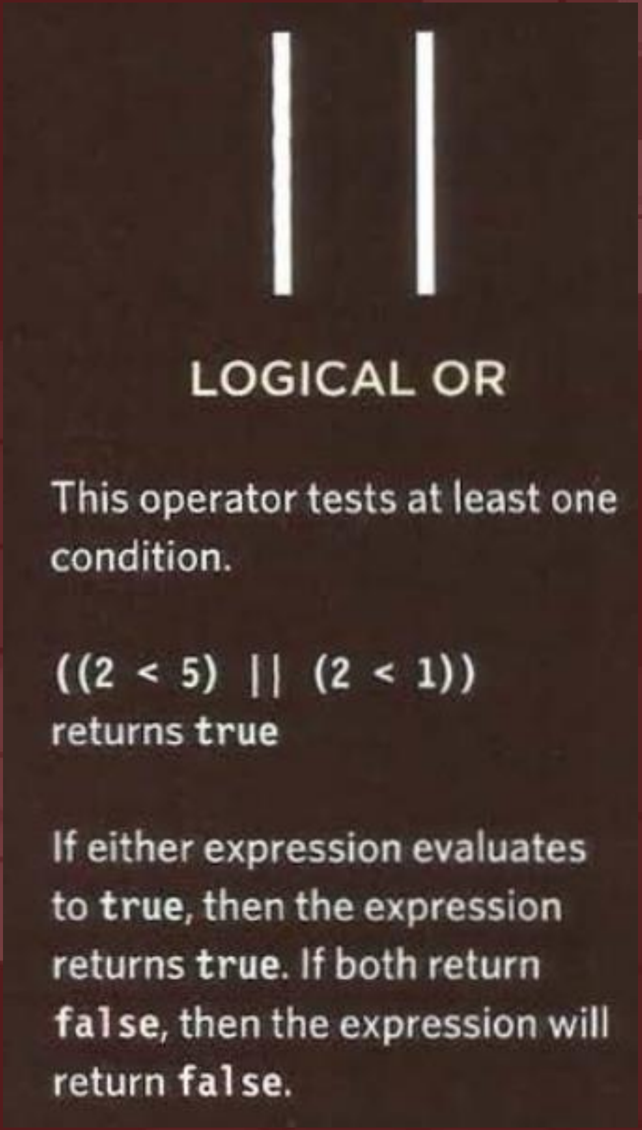
&&

LOGICAL AND

This operator tests more than one condition.

`((2 < 5) && (3 >= 2))`
returns `true`

If both expressions evaluate to `true` then the expression returns `true`. If just one of these returns `false`, then the expression will return `false`.



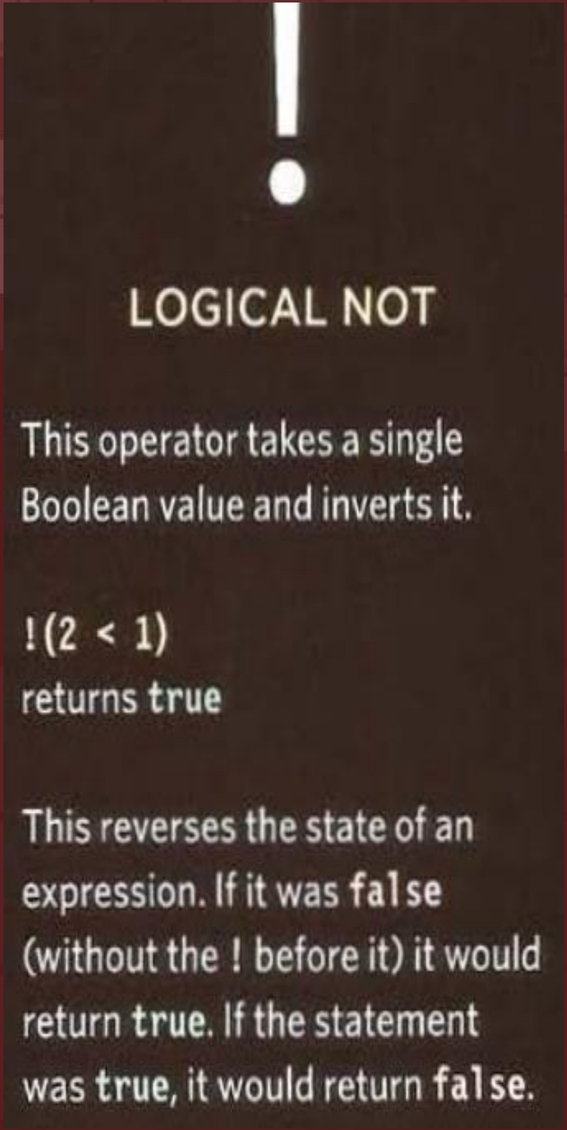
||

LOGICAL OR

This operator tests at least one condition.

`((2 < 5) || (2 < 1))`
returns `true`

If either expression evaluates to `true`, then the expression returns `true`. If both return `false`, then the expression will return `false`.



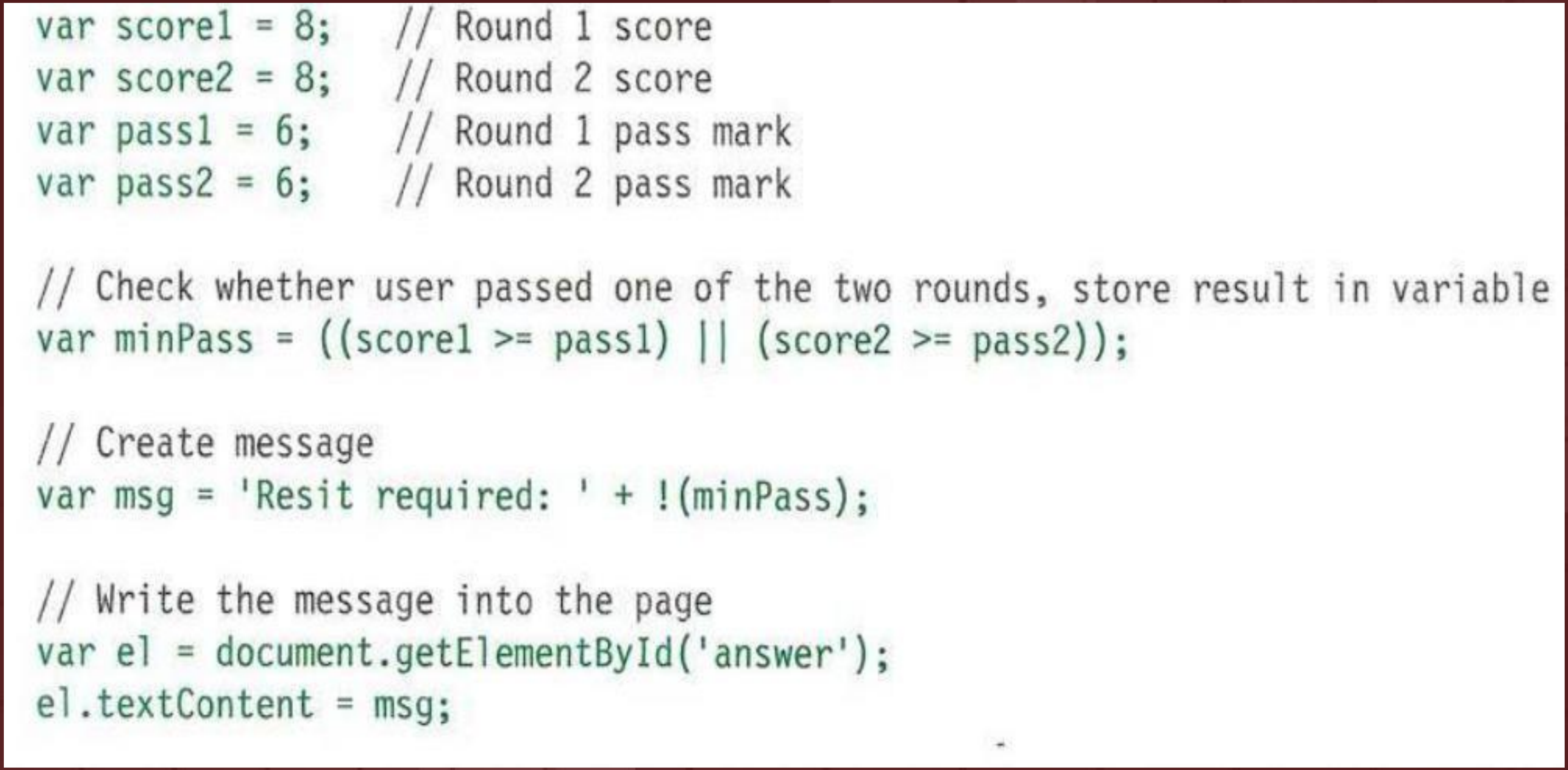
!

LOGICAL NOT

This operator takes a single Boolean value and inverts it.

`!(2 < 1)`
returns `true`

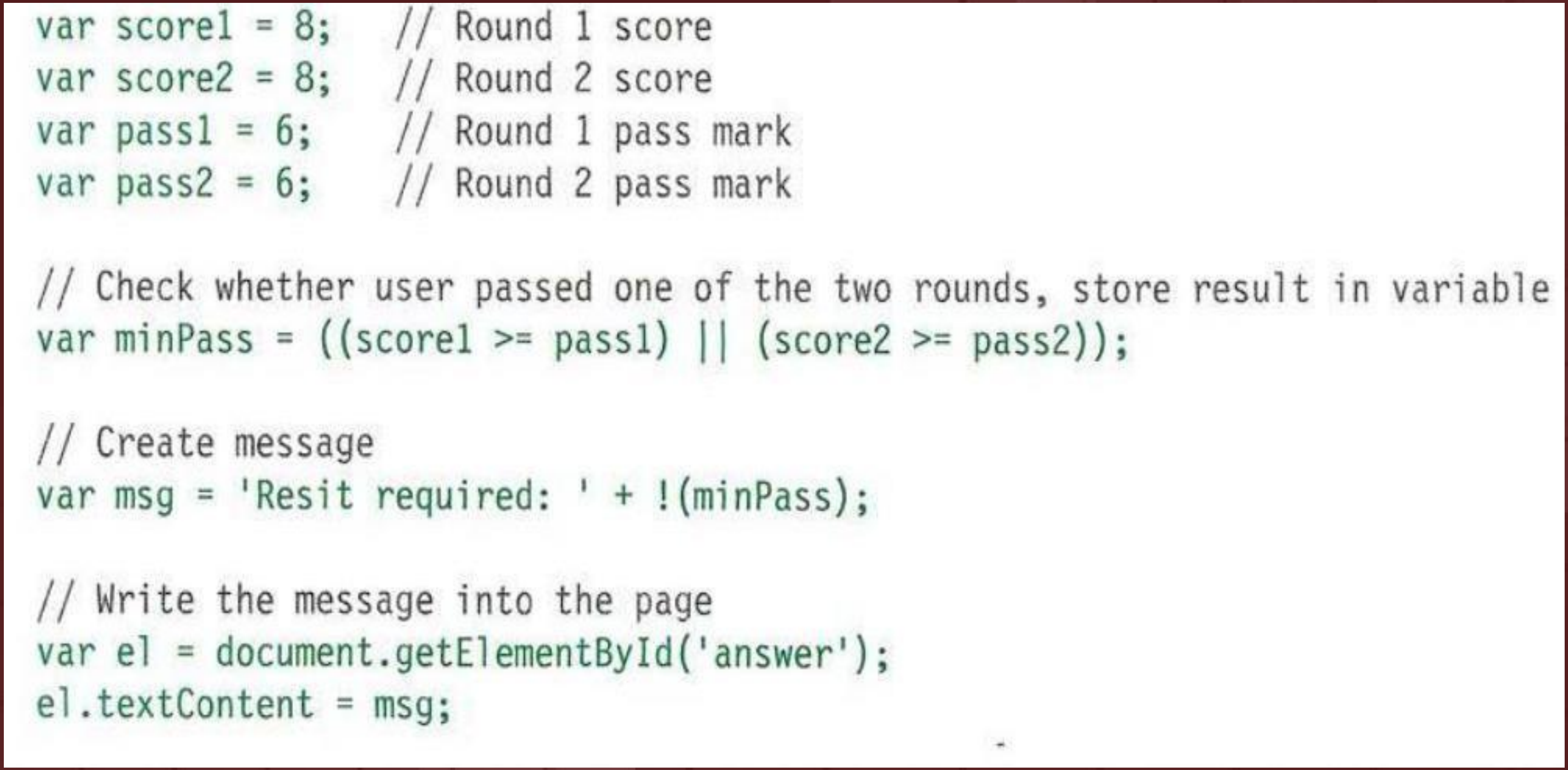
This reverses the state of an expression. If it was `false` (without the ! before it) it would return `true`. If the statement was `true`, it would return `false`.



```
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 one of the two rounds, store result in variable
var minPass = ((score1 >= pass1) || (score2 >= pass2));

// Create message
var msg = 'Resit required: ' + !(minPass);

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

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.



```
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 one of the two rounds, store result in variable
var minPass = ((score1 >= pass1) || (score2 >= pass2));

// Create message
var msg = 'Resit required: ' + !(minPass);

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

IF Statements:

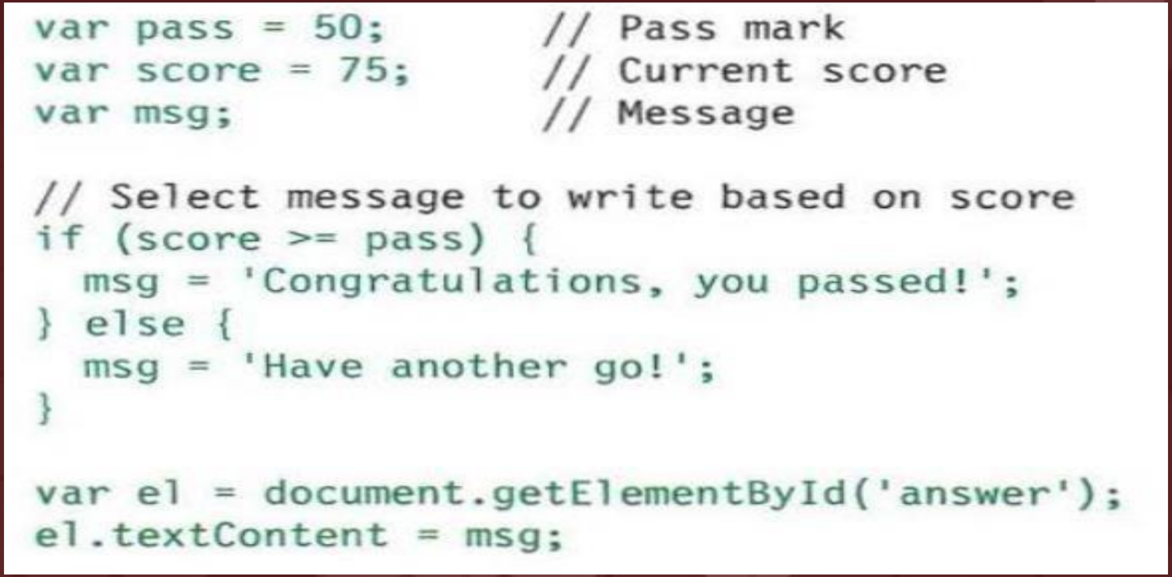
The if statement evaluates (or checks) a condition. If the condition evaluates to true , any statements in the subsequent code block are executed.

IF … Else Statements:

IF .. Else statement checks a condition.

If it resolved to true the first code block is executed

If the condition resolved to false the second code block is executed



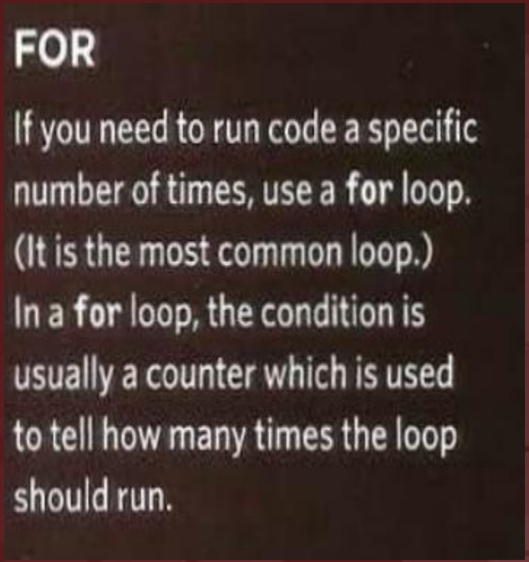
```
var pass = 50;          // Pass mark
var score = 75;         // Current score
var msg;                // Message

// Select message to write based on score
if (score >= pass) {
  msg = 'Congratulations, you passed!';
} else {
  msg = 'Have another go!';
}

var el = document.getElementById('answer');
el.textContent = msg;
```

JavaScript can convert data types behind the scenes to complete an operation. This is known as type coercion. For example, a string 'l ' could be converted to a number 1 in the following expression:(' 1' > 0). As a result, the above expression would evaluate to true.

fppt.com

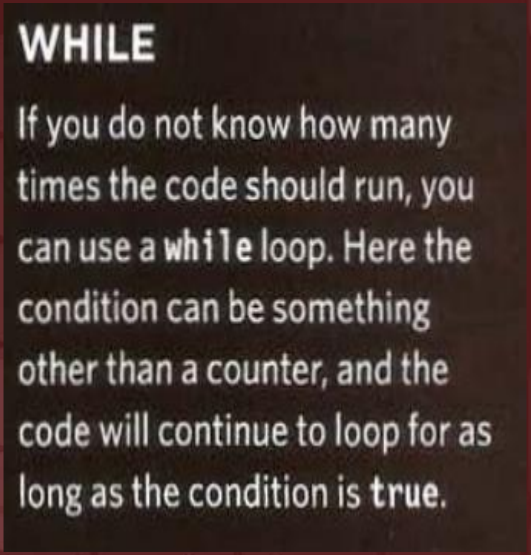
LOOPS:

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, a code block will run gain. It repeats until the condition returns false. There are three types of loops.



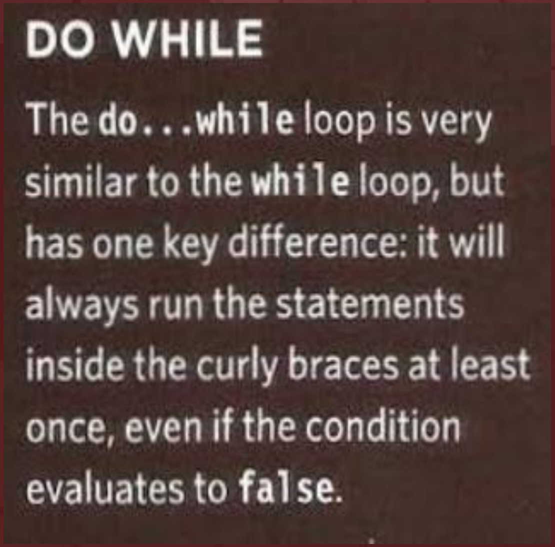
FOR

If you need to run code a specific number of times, use a **for** loop. (It is the most common loop.) In a **for** loop, the condition is usually a counter which is used to tell how many times the loop should run.



WHILE

If you do not know how many times the code should run, you can use a **while** loop. Here the condition can be something other than a counter, and the code will continue to loop for as long as the condition is **true**.



DO WHILE

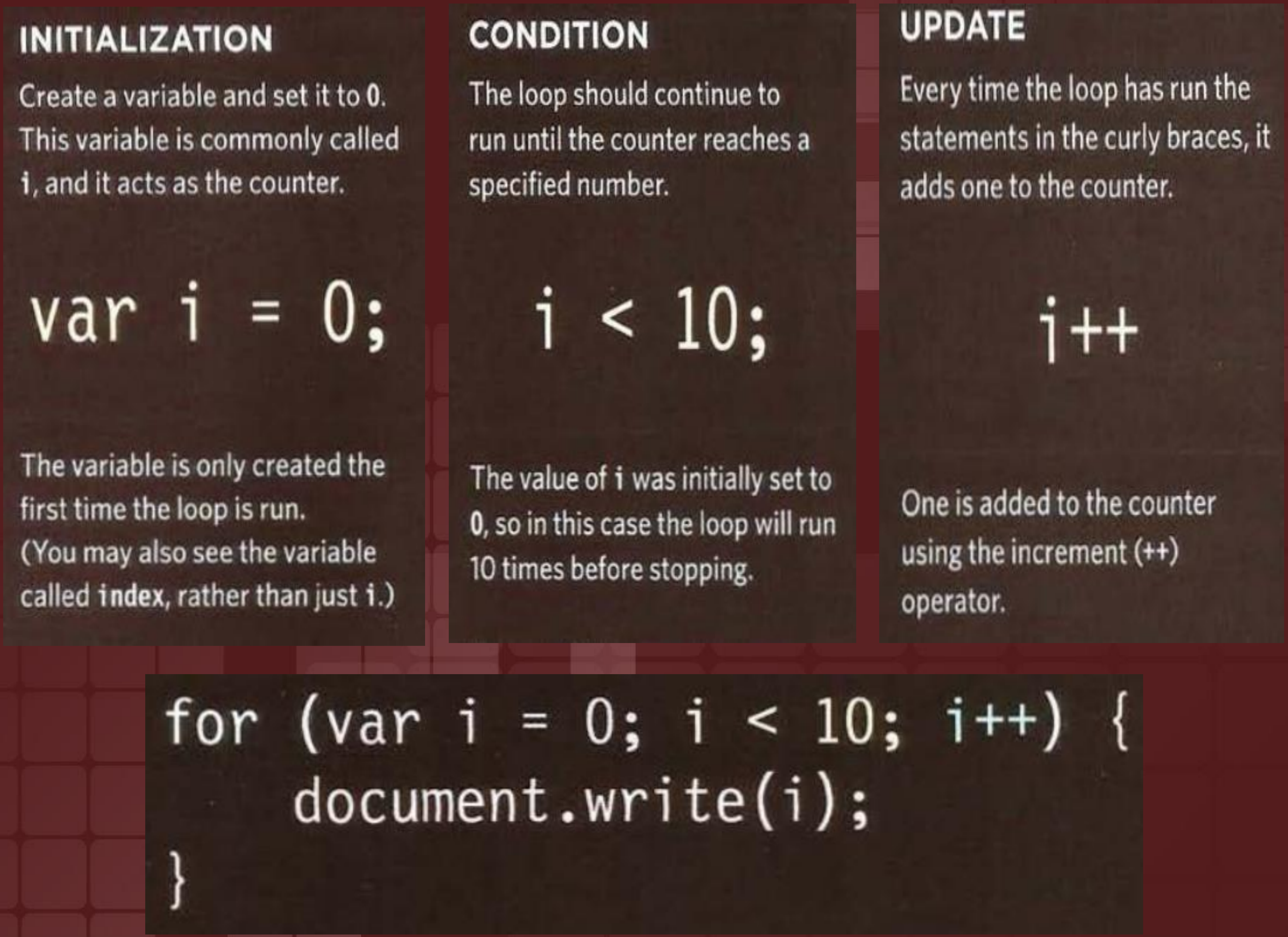
The **do...while** loop is very similar to the **while** loop, but has one key difference: it will always run the statements inside the curly braces at least once, even if the condition evaluates to **false**.

For Loop:

For loop uses counter as a condition

This instruct the code to run specified number of times

Here you can see the condition made up of three statements



INITIALIZATION

Create a variable and set it to **0**. This variable is commonly called **i**, and it acts as the counter.

```
var i = 0;
```

The variable is only created the first time the loop is run. (You may also see the variable called **index**, rather than just **i**.)

CONDITION

The loop should continue to run until the counter reaches a specified number.

```
i < 10;
```

The value of **i** was initially set to **0**, so in this case the loop will run 10 times before stopping.

UPDATE

Every time the loop has run the statements in the curly braces, it adds one to the counter.

```
i++
```

One is added to the counter using the increment (++) operator.

```
for (var i = 0; i < 10; i++) {
    document.write(i);
}
```

KEY LOOP CONCEPTS

Here are three points to consider when you are working 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

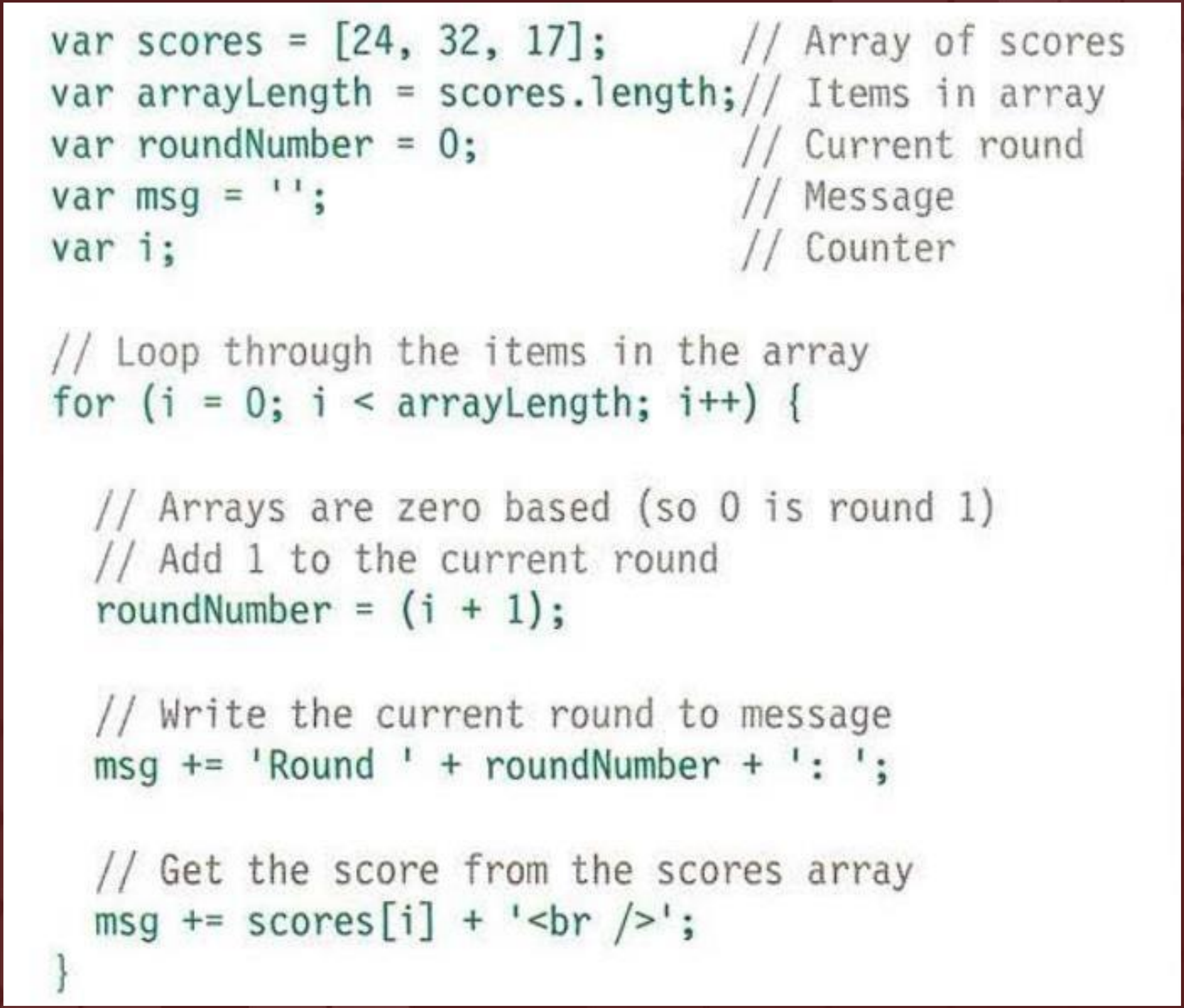
Continue :

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).

LOOPS & ARRAYS

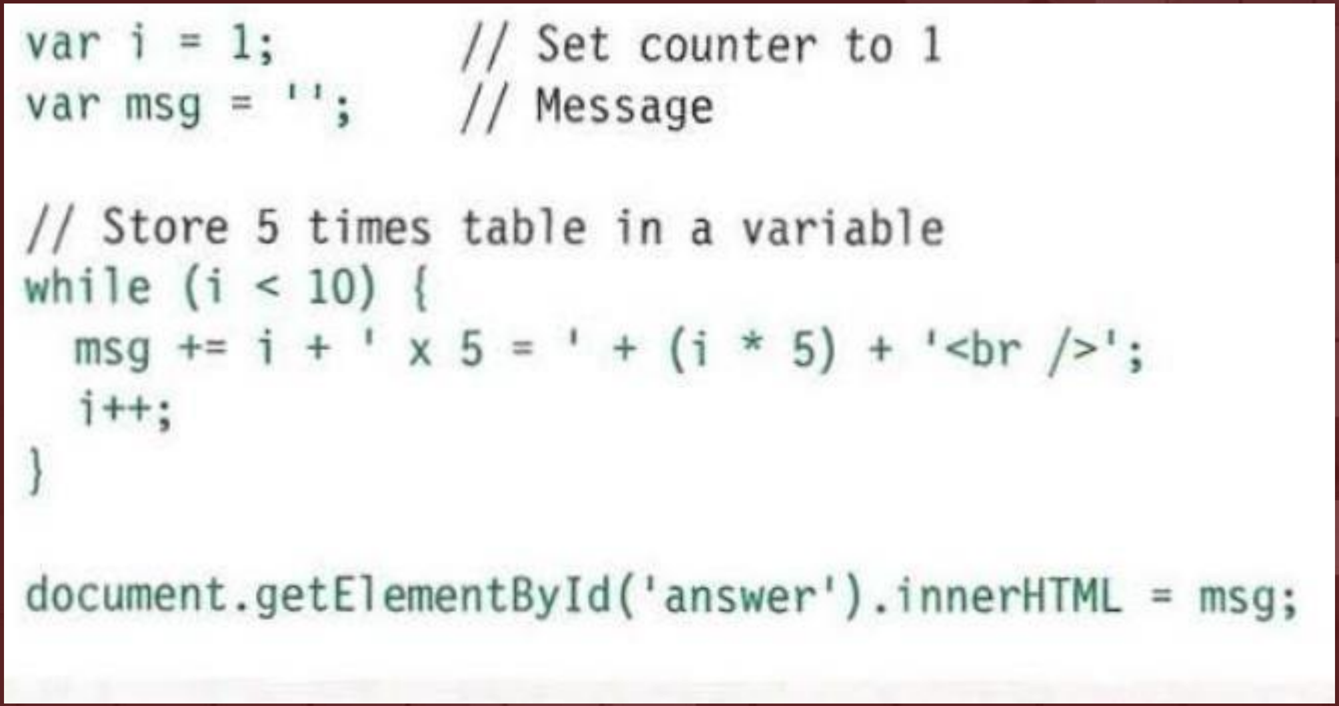
Loops are very helpful when dealing with arrays if you want to run the same code for each item in the array. For example, you might want to write the value of each item stored in an array into the page.

USING FOR LOOPS



```
var scores = [24, 32, 17];         // Array of scores
var arrayLength = scores.length;// Items in array
var roundNumber = 0;               // Current round
var msg = '';                      // Message
var i;                             // Counter

// Loop through the items in the array
for (i = 0; i < arrayLength; i++) {

  // Arrays are zero based (so 0 is round 1)
  // Add 1 to the current round
  roundNumber = (i + 1);

  // Write the current round to message
  msg += 'Round ' + roundNumber + ': ';

  // Get the score from the scores array
  msg += scores[i] + '<br />';
}
```

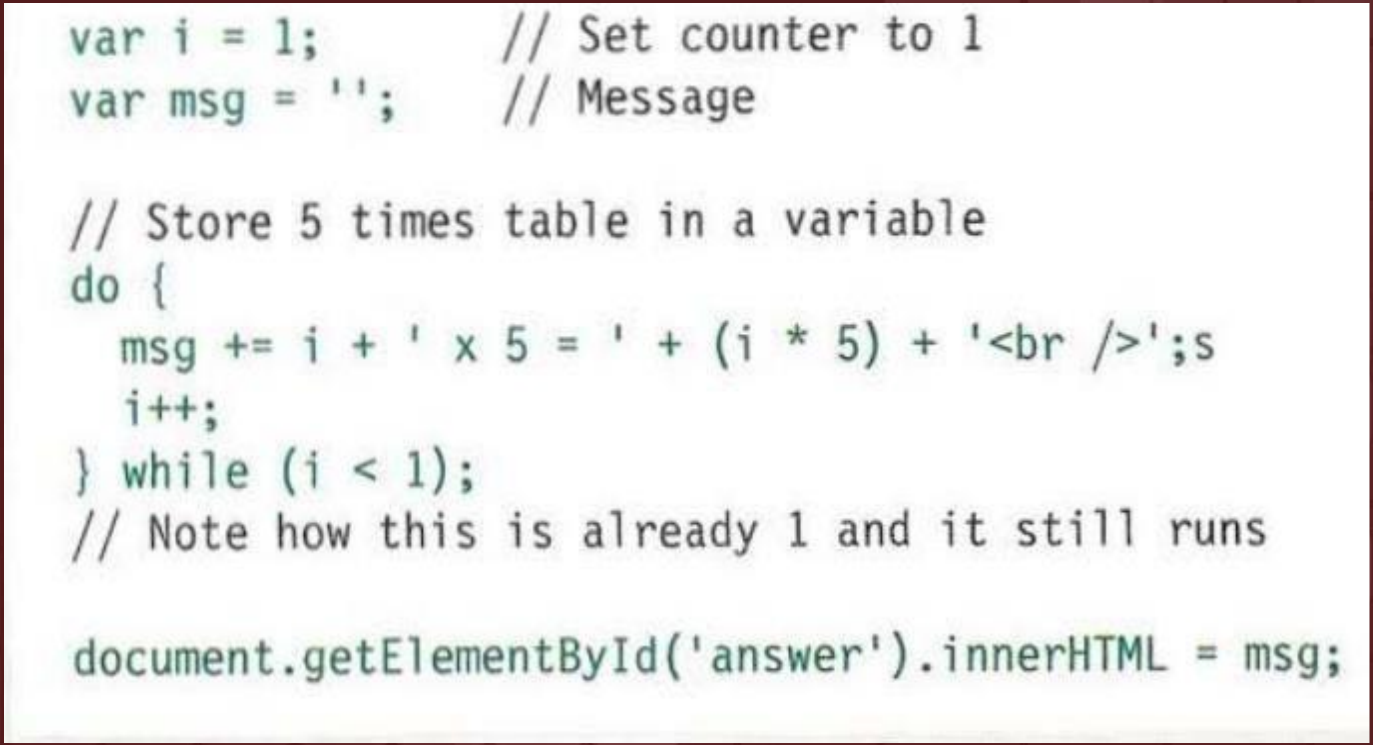
USING WHILE LOOPS



```
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;
```

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.

USING DO WHILE LOOPS



```
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 />';s
  i++;
} while (i < 1);
// Note how this is already 1 and it still runs

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

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.

SUMMARY

Conditional statements allow your code to make decisions about what to do next.

Comparison operators (===, ! ==, ==, ! =, <, >, <=, =>) are used to compare two operands.

Logical operators allow you to combine more than one set of comparison operators.

if … else statements allow you to run one set of code if a condition is true, and another if it is false.

switch statements allow you to compare a value against possible outcomes (and also provides a default option if none match).

SUMMARY

Data types can be coerced from one type to another.
All values evaluate to either truthy or falsy.

There are three types of loop: for, while, and
do … while. Each repeats a set of statements.

Next Week

Document Object Model (DOM)

Questions ?