
Foundation Coding - Week 2: Basics



Conditionals - if/else

```
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  
}
```



Conditionals - else if

```
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  
}
```



Switch


Another logic pattern that will look for a true match to a condition. If a match/case is found the code will run.

In this example the *new Date().getDay()* code will produce a number between 0 - 6. The switch is looking for a match and producing a day.

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

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {  
  case 6:  
    text = "Today is Saturday";  
    break;  
  case 0:  
    text = "Today is Sunday";  
    break;  
  default:  
    text = "Looking forward to the Weekend";  
}
```



Comparison Operators

COMPARISON OPERATORS: EVALUATING CONDITIONS

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

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

`'Hello' == 'Goodbye'` returns **false**
because they are *not* the same string.
`'Hello' == 'Hello'` returns **true**
because they *are* the same string.

It is usually preferable to use the strict method:

!=

IS NOT EQUAL TO

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

`'Hello' != 'Goodbye'` returns **true**
because they are *not* the same string.
`'Hello' != 'Hello'` returns **false**
because they *are* the same string.

It is usually preferable to use the strict method:

===

STRICT EQUAL TO

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

`'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.

!==

STRICT NOT EQUAL TO

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

`'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.

Comparison Operators

>

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

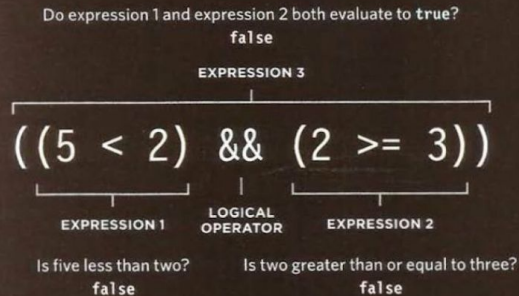
<=

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


Comparison Operators



In this one line of code are three expressions, each of which will resolve to the value **true** or **false**.

The expressions on the left and the right both use comparison operators, and both return **false**.

The third expression uses a logical operator (rather than a comparison operator). The logical AND operator checks to see whether both expressions on either side of it return **true** (in this case they do not, so it evaluates to **false**).



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**.

```
true && true  returns true  
true && false returns false  
false && true  returns false  
false && false returns 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**.

```
true || true  returns true  
true || false returns true  
false || true  returns true  
false || false returns 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**.

```
!true  returns false  
!false returns true
```




DOM Queries

Elements can be selected by many methods. 2 popular methods are *getElementById* and *querySelector*.

The *getElementById* method will only select elements with an id match.

The *querySelector* method can select elements by id, class or element identification. This method will select the first match found.

```
// <p id="one">A paragraph...</p>
// <p id="two">Another paragraph...</p>
// <p class="three">Another paragraph...</p>

var el = document.getElementById('one');
// -----
var eltwo = document.querySelector('#two');
var elthree = document.querySelector('.three');
// This will select the first paragraph found in the html/DOM
var firstParagraph = document.querySelector('p');
```

References:

https://www.w3schools.com/js/js_if_else.asp

https://www.w3schools.com/jsref/met_document_queryselector.asp

https://www.w3schools.com/js/js_comparisons.asp

Images referenced from Duckett:

<http://javascriptbook.com/>
