

Conditional and Logic Statement

true and false values

In JavaScript, the Boolean data type represents one of two values: true or false. Those values can also be assigned to a variable as:

```
var day = true;
var night = false;
```

The Boolean variables are later used in the program to test or compare if a given condition is true or false.

```
if (day){
  console.log("The statement is: ", day);
}
```

```
> var day = true;
< undefined
> var night= false;
< undefined
> if (day){console.log("The statement is: ", day);}
  The statement is: true
< undefined
> |
```

A lot of programming is about testing for conditions, asking if some condition is met, then do this, otherwise do something else. This is done using conditional statements and logic operators.

Boolean operator	Purpose
==	equality
===	Exact equal to
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
&&	And
	Or
!	Not

Conditional if statement

if statement testes if a condition is true, then do something. It is also known as an ON/OFF switch condition.

Syntax

```
if (testing condition) { do something }
```

Here, the parenthesis wrap the condition, or conditions, and the curly braces wrap what happens if the condition is met. This is known as the code block. You could technically write this all on one line, but that would be hard to read for the humans who will be looking at your code, so you can write the first curly brace on one line, then the code block inside in the next line.

```
if (testing condition)  
    { do something }
```

Example) Use if statement to check a person mood

```
var goodMood = true;  
var gotSleep = true;  
  
if (goodMood && gotSleep){  
    console.log("Today is a good day!");  
}
```

```
console.log("I am moody!");
```

```
Today is a good day!
```

```
var goodMood = true;  
var gotSleep = false;  
if (goodMood && gotSleep){  
    console.log("Today is a good day!");  
}
```

```
console.log("I am moody!");
```

```
I am moody!
```

```
var goodMood = true;  
var gotSleep = false;  
var eat = false  
if (goodMood || gotSleep || eat){  
    console.log("Today is a good day!");  
}  
console.log("I am moody!");
```

```
Today is a good day!
```

```
I am moody!
```

if – else condition

If you want to create an **if/else** statement, where you perform a separate specific action **if** the condition is not met as well, you add **else**, and a new set of curly braces after the **if** statement. If some condition is true, do something, **else**, if the condition it's not true, then check something **else**. If statements require conditions to work, so we need some logic operators.

Example) Check if two numbers are equal

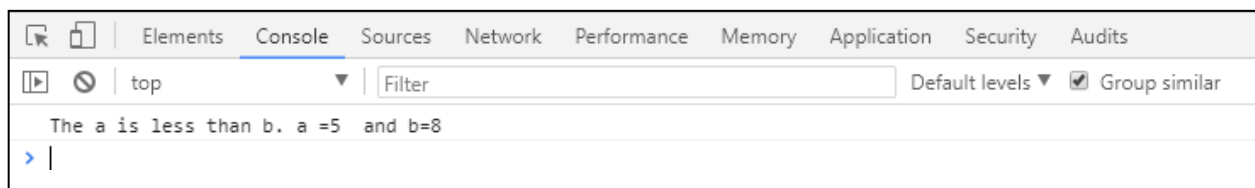
```
var a = 5;
var b = 4;
var equalNum;

if (a==b)
    {equalNum = true; }
else
    {equalNum = false; }

console.log("The numbers match: " + equalNum);
```

```
var a = 5;
var b = 8;

if (a===b){
    console.log("The numbers are equal");
}
else if (a>b) {
    console.log("a is greater than b");
}
else if (b>a) {
    console.log("The a is less than b");
}
else {
    console.log("Invalid Entry");
}
```



Example) use an **if, else if** statement to check which color is selected from an array

```
var colors = ['red','blue','green','yellow','orange'];
var selectColor = colors[2];
if (selectColor=='red'){
    console.log('The selected color is red');
}
else if (selectColor=='blue'){
    console.log('The selected color is blue');
}
else if (selectColor=='green'){
    console.log('The selected is green');
}
else if (selectColor=='yellow'){
    console.log('The selected color is yellow');
}
else if (selectColor=='orange'){
    console.log('The selected color is orange');
}
else{
    console.log('Color is not in the list!');
}
```

Result → **The selected color is green**

Since JavaScript allows data type collision, meaning that it can modify data to produce the same resulting data type, then it uses a triple equal sign === Boolean operator, to identify exact equal data values. For example, if we want to compare if number 5 and string value 5 are equal values, by using double equal signs, JavaScript will apply data type collision and will treat both value as the same value:

```
var num = 10;
var otherNum = "10";
if(num>otherNum){
    console.log("Number ", num, "is greater than ",otherNum)
}
else if(num==otherNum){
    console.log("Number ", num, "is equal to ",otherNum)
}
else if(num<otherNum){
    console.log("Number ", num, "is less than ",otherNum)
}
else{
    console.log("Different data values");
}
```

Result in the browser:

Number 10 is equal to 10

Now, if we replace the double equal signs with the triple equal signs, then JavaScript will check for the exact the same type and value.

```
var num = 10;
var otherNum = "10";
if(num>otherNum){
  console.log("Number ", num, "is greater than ",otherNum)
}
else if(num===otherNum){
  console.log("Number ", num, "is equal to ",otherNum)
}
else if(num<otherNum){
  console.log("Number ", num, "is less than ",otherNum)
}
else{
  console.log("Different data values");
}
```

Result in browser **Different data values**

Switch

Switch statements works the same as if, else if, else statement, with the difference that we select among different cases.

Example) use switch statement to check which color is selected from an array

```
var colors = ['red','blue','green','yellow','orange'];
var selectColor = colors[2];

switch(selectColor){
  case "red":
    console.log('The selected color is red');
    break;
  case "blue":
    console.log('The selected color is blue');
    break;
  case "green":
    console.log('The selected color is green');
    break;
  case "yellow":
    console.log('The selected color is yellow');
    break;
  case "orange":
    console.log('The selected color is orange');
    break;
  default:
    console.log('Color is not in the list!');
}
```

Result in browser → **The selected color is green**

Loops

In the real world, that is rarely how things work and, in many cases, we specifically want our code to do things more than once, usually as many times as is necessary and sometimes even endlessly. For this, we have loops. Loops are a vital part of all programming languages and will play a vital role in most JavaScript code. At their core, loops are simple. We create some sort of loop condition and as long as this condition holds or true, the loop will keep running.

for loop

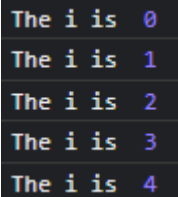
for loop runs the statement as long as the condition is true.

Syntax

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Example) use a for loop to display an increasing counter from 0 to 4

```
for (let i = 0; i < 5; i++) {  
    console.log( "The current i is ", i);  
}
```

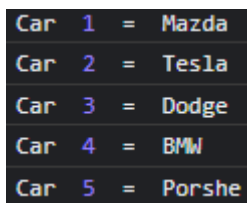


```
The i is 0  
The i is 1  
The i is 2  
The i is 3  
The i is 4
```

for loop in an array

for loop is very useful to loop to each values in an array. When we work with for loop in an array, we have to keep in mind that zero is the most common initial value because it is the initial index value in an array.

```
var cars = ['Mazda', 'Tesla', 'Dodge', 'BMW', 'Porsche'];  
for(var i=0; i<cars.length; i++){  
    console.log('Car ', i+1, ' = ', cars[i]);  
}
```



```
Car 1 = Mazda  
Car 2 = Tesla  
Car 3 = Dodge  
Car 4 = BMW  
Car 5 = Porsche
```

There is also a specific statement in a for loop that works with array, which is the **for... of** statement. The **of** statement in a for loop will loop to each item in the list

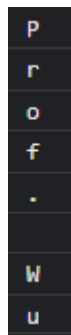
```
var cars = ['Mazda', 'Tesla', 'Dodge', 'BMW', 'Porsche'];
for(var eachItem of cars){
    console.log(eachItem);
}
```



for loop in a string

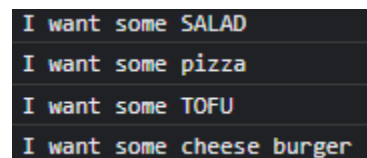
We can also use **for** loop to go to each of the character in an string:

```
var myString = "Prof. Wu";
for(eachLetter of myString){
    console.log(eachLetter);
}
```



Example) Nest statements: Display a message I want some _____ with each food within an array. All even foods will display upper cases.

```
var foods = ['salad', 'pizza', 'tofu', 'cheese burger'];
for(var i=0; i<foods.length;i++){
    if (i%2 === 0){
        x=foods[i].toUpperCase();
        console.log('I want some ' + x);
    }
    else{
        console.log('I want some ' + foods[i]);
    }
}
```



while loop

The for loop assumes you know how many times you want to loop to run but sometimes you just want to run the loop until some condition changes. In that case, you can use a while loop instead.

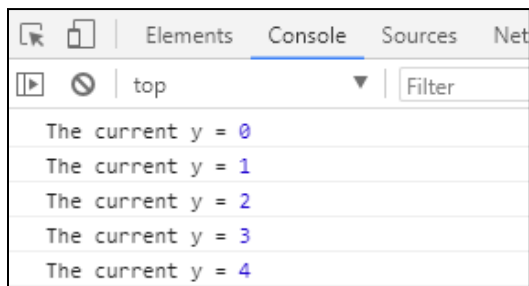
The while loop allows us to create more advanced functions inside the core block and run the loop as long as these or other external conditions are true.

Syntax

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

Example) use **while** loop to display number from 0 to 4

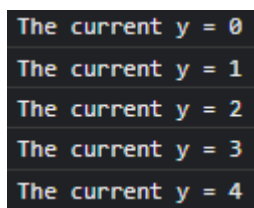
```
var y = 0; while (y<5) {  
    console.log("The current y =",y); y++;  
}
```



Template literals

Using backticks `` in JavaScript along with `${}` around the expressions allows to embed information in the `${}` into the string.

```
var y = 0; while (y<5) {  
    console.log(`The current y = ${y}`); y++;  
};
```



Bibliography

- Duckett, J. (2016). *HTML and CSS Design and build websites*. Indianapolis: John Wiley and Sons Inc.
- Duckett, J. (2016). *JavaScript and JQuery: interactive front-end developer*. Indianapolis: John Wiley and Sons Inc.
- *HTML, CSS, JavaScript, JQuery, and Bootstrap*. (2017, June). Retrieved from w3schools: www.w3schools.com
- Some material compiled from www.lynda.com (May 2018), www.w3schools.com (2020), www.coursera.org (2020)

IMPORTANT NOTE

The materials used in this manual have the author's rights and are for educational use only.