

Conditional and Logic Statement

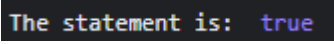
true and false values

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

```
let day = true;
let 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);
}
```

The display in the console → 

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

Character or string comparison with **greater than >** or **less than <** with compare the character Unicode value

```
> "peter">"Peter"
< true
> "p">"P"
< true
> "p"<"P"
< false
```

== double equals checks for equality of value, but not equality of type. It basically treats both values as the same type and then compares them. This can be very problematic.

```
> 20 == 20
< true
> "20" == 20
< true

> 0 == false
< true
> 1 == true
< true
> null == undefined
< true
```

=== triple equals sign checks for equality of type and value

```
> "20"===20
< false
> 20===20
< true
> 1===true
< false
> 0===false
< false
> null===undefined
< false
```

!= not equal to compares if the values, and not the type, that are not equal to each other

```
> 5 != 5
< false
> 5 != '5'
< false
```

!== exactly not equal to compares if the values and the types are not equal to each other

```
> 5 !== 5
< false
> 5 !== '5'
< true
```

For comparison, it is recommended to use triple equal **===** and exactly equal to **!==**

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

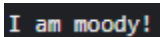
```
if (testing condition) { do something }
```

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 }
```

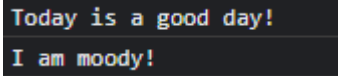
Example) Use if statement to check a person mood

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

the console displays → 

If we change the conditional to triple equal sign, we are basically comparing if the two logic statements are true.

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

The console displays → 

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

```
let a = 5, b = 4, equalNum;

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

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

The console displays ➔ `The numbers match: false`

if – else if - else condition

if-else if-else condition test more than two statement.

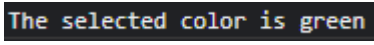
```
let num1 = 5, num2 = 8;

if (num1===num2){
  console.log("The numbers are equal");
}
else if (num1>num2) {
  console.log("number 1 is greater than number 2");
}
else if (num2>num1) {
  console.log("The number 1 is less than number 2");
}
else {
  console.log("Invalid Entry");
}
```

The console displays ➔ `The number 1 is less than number 2`

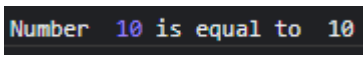
Example) use an **if, else if** statement to check which color is selected from an array

```
let colors = ['red','blue','green','yellow','orange'];
let 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!');
}
```

The console displays → 

Remember that 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:

```
let num = 10;
let 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");
}
```

The console displays → 

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.

```
let num = 10;
let 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");
}
```

The console displays → 

***isNaN()* function**

isNaN(), is Not a Number, is a function that can check if a value is not a number. If the variable is not a number, it returns true, else, if the variable is a number, it returns false.

Example) Prompt a window and ask the user to enter a number or a string. Prompt at the console the result

```
let user = prompt('Please enter a number or string: ');
checkUser = parseInt(user);
if (isNaN(checkUser)){
    console.log(`${user} is a string`);
}
else{console.log(`${checkUser} is a number`);}
```

Nesting conditions

Nesting condition is when we have a condition inside of a condition. It basically check another condition of a true condition.

Example) Create a JS code, using if-else statement, to check if a password is at least six characters not counting the space. If the password has more than six characters but it also has space, then the program will display an error message.

```
let password= prompt("Enter a password");
if (password.length>=6){
    console.log("Password has 6+ characters!");
    if (password.indexOf(' ')===-1){
        console.log("Great! Password has no space!");
    }
    else{
        console.log("Warning! Password can't have space! ");
    }
}
else{console.log("Password has less than 6 characters! ");}
```

Running the program, if user types *Peter Pan* and click OK, the console will display:

This page says

Enter a password

OK Cancel

```
Password has 6+ characters!
Warning! Password can't have space!
```

If user types *Peter* and click OK, the console will display:

This page says

Enter a password

OK Cancel

```
Password has less than 6 characters!
```

If user types *PeterPan* and click OK, the console will display:

This page says

Enter a password

OK Cancel

```
Password has 6+ characters!
Great! Password has no space!
```

True and false values

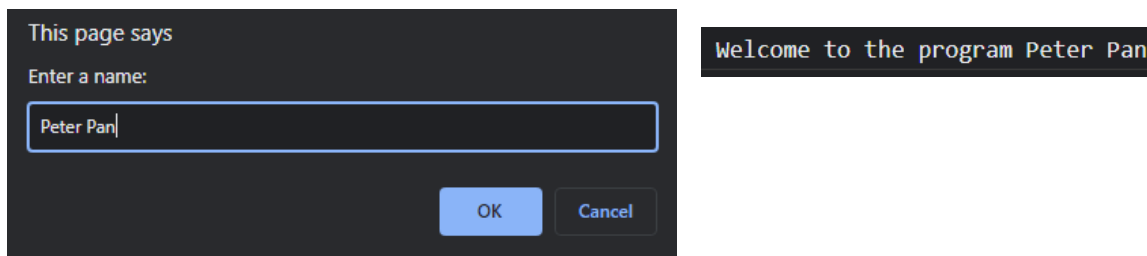
Some statement or values are assigned to be true or false depending on the condition.

Example) Write a JS code that asks the user to enter a name. If the user enters a name, it will display a welcome message with the entered name. Otherwise, if the user does not enter a name, it will display that no name was entered.

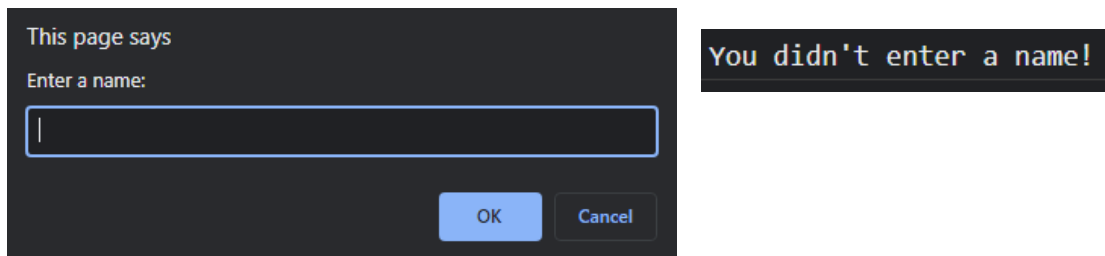
```
let name = prompt("Enter a name: ");

if (name){console.log(`Welcome to the program ${name}`)}
else{console.log(`You didn't enter a name!`)}
```

If user types Peter Pan and click OK, the console will display:



If user doesn't type anything and click OK, the console will display:



A null, empty, undefined, NaN, and 0 are inherent to be false. Otherwise, any string, numbers, or even a space are inherent to be true.

Logical operators

AND operator &&

In a chain of statements, compares if all statements are true for the entire statement to be true. In another words, the statement is true if all the conditions are true.

```
> true && false && true
< false
```

```
true && true && true
true
```


Example) modify the example in the nesting if-else statement using AND operation

```
let password= prompt("Enter a password");
if (password.length>=6 && password.indexOf(' ')===-1){
    console.log(`${password} is valid password`);
}
else{
    console.log(`WARNING! ${password} is incorrect password format`);}
}
```

This page says

Enter a password

OK Cancel

If the user types *Peter Pan* and click OK, the console will display:

WARNING! Peter Pan is incorrect password format

This page says

Enter a password

OK Cancel

If the user types *PeterPan* and click OK, the console will display:

GREAT! PeterPan is valid password

This page says

Enter a password

OK Cancel

If the user types *mary* and click OK, the console will display:

WARNING! mary is incorrect password format

Example) type a mystery constant that will make the console to display *YOU GOT IT!!!*

```
const mystery = 'Pat007James';

if(mystery[0] === 'P' && mystery.length > 5 && mystery.indexOf('7') !== -1){
    console.log("YOU GOT IT!!!");
}
else{console.log("Wrong Guess!");}
```

Console will display → YOU GOT IT!!!

OR // operator

From a chain of statements, if one statement is true, the entire statement is true.

```
> true || true || false
< true
> false || false || true
< true
> false || true || false
< true
> false || false || false
< false
> true || true || true
< true
```

Example) write a JS code that asks the user to rate a book as:

1. Excellent
2. Good/Average
3. Bored

If the user select “Bored”, the program will prompt another window asking why the book was rated as “Bored”

```
let rate = prompt('Enter a rate for the book "The book". \n 1) Excellent \n 2) Good\Average \n 3) Bored \nSelect a number from 1 to 3.')

if(rate==='3'){
  let comment = prompt('We are sorry that you find "The Book" boring. Please comment what we can do to improve your experience with "The Book" ?');
  console.log(`Thank you for your feedback: ${comment}`)
}
else if (rate ==='1' || rate ==='2'){
  console.log('Thank you for choosing "The Book".')
}
else{
  console.log('INVALID RATE!')
}
```

Combining logical operators

We can also combine logical operators in one statement. If you combine logical operators, AND operator has precedent over OR operator, meaning that a JS code always checks on the AND operator before the OR operator.

Example) Tickets to a park charge the following fee:

Age	Price
0 to 7	FREE
8 to 17	\$ 5.00
18 to 64	\$ 8.00
65 and up	FREE

Create a JS code that will display the price according to the age or status.

```
let age = 18;

if (age<=7 || age>=65){console.log("FREE ticket")}
else if(age<=17 ){console.log("$5.00 ticket per person")}
else if(age<65){console.log("$8.00 ticket per person")}
else{console.log("INVALID AGE!")}
```

If you want to control the negative ages, we can add an AND operators for ages between 0 and 7. Since in JS, an AND operator is checked first, the program will check the ages in between of the && operators first.

```
let age = -20;

if (age>=0 && age<=7 || age>=65){console.log("FREE ticket")}
else if(age<=17 && age>7 ){console.log("$5.00 ticket per person")}
else if(age<65 && age>17){console.log("$8.00 ticket per person")}
else{console.log("INVALID AGE!")}
```

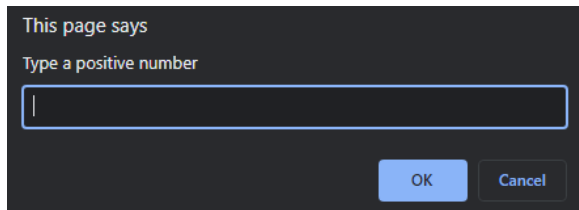
NOT ! operator

NOT operator returns a true if an expression is false.

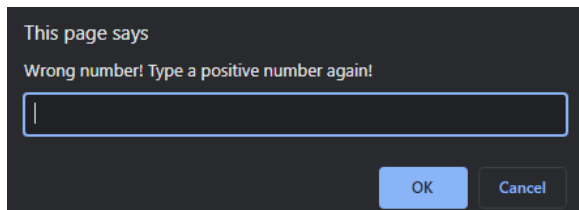
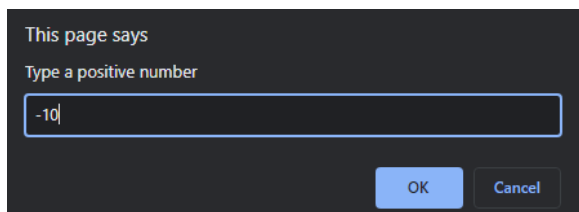
```
> !(7 === '7')
< true
> !(9 === 6)
< true
> !(8 === 8)
< false
```

Example) write a JS code that will ask the user to type a positive number in the prompt dialog. If the user enters a negative number, then the program will prompt again dialog asking to enter another number. This program is good only for one check point.

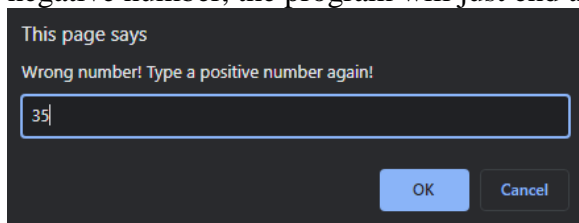
```
let msg = parseInt(prompt("Type a positive number "));
if (msg < 0){
    msg = parseInt(prompt("Wrong number! Type a positive number again! "));
}
console.log(`Entered number: ${msg}`)
```



If user types -10 and click OK, another dialog box appears asking to enter another number...



Remember that this program is good for one check point. Therefore, if the user enters a positive or negative number, the program will just end the if statement and display a message in the console.



```
Entered number: 35
```

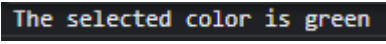
Switch

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

```
let colors = ['red', 'blue', 'green', 'yellow', 'orange'];
let 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 console → 

You can also check multiple cases with the same output:

```
let selectColor = 'BLUE';

switch(selectColor){
  case "red": case "RED":
    console.log('The selected color is red');
    break;
  case "blue": case "BLUE":
    console.log('The selected color is blue');
    break;
}
```

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.