Conditionals and Loops

The if Statement

Very often when you write code, you want to perform different actions based on different conditions.
You can do this by using **conditional statements** in your code.

Use **if** to specify a block of code that will be executed if a specified condition is true.

```
if (condition) {
  statements
}
```

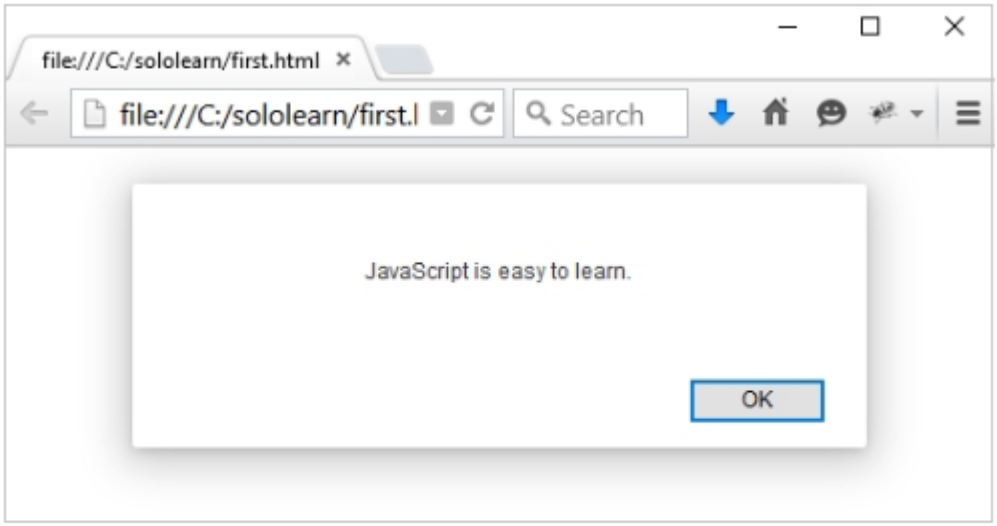
The statements will be executed only if the specified condition is **true**.

Example:

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 < myNum2) {
  alert("JavaScript is easy to learn.");
}
```

Try It Yourself

Result:



As seen in the picture above, the JavaScript **alert()** method is used to generate a popup alert box that contains the information provided in parentheses.

The if Statement

This is another example of a **false** conditional statement.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
  alert("JavaScript is easy to learn.");
}
```

Try It Yourself

As the condition evaluates to false, the alert statement is skipped and the program continues with the line after the if statement's closing curly brace.

Note that **if** is in lowercase letters. Uppercase letters (If or IF) will generate an error.

The else Statement

Use the **else** statement to specify a block of code that will execute if the condition is **false**.

```
if (expression) {
  // executed if condition is true
}
else {
  // executed if condition is false
}
```

You can skip curly braces if your code under the condition contains only one command.

The else Statement

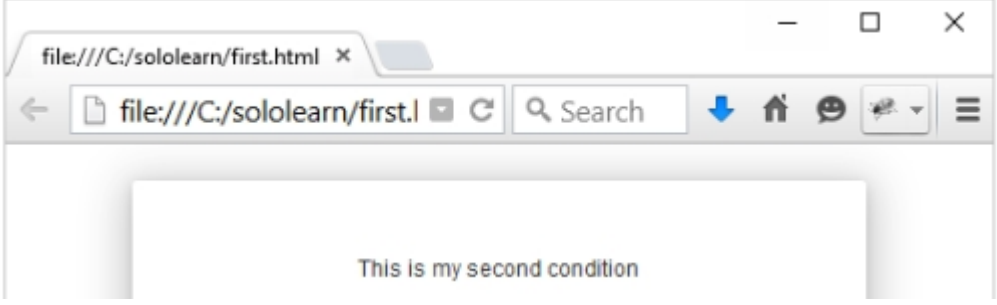
The example below demonstrates the use of an **if...else** statement.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
  alert("This is my first condition");
}
else {
  alert("This is my second condition");
}
```

Try It Yourself

The above example says:
- **If** *myNum1* is greater than *myNum2*, alert "This is my first condition";
- **Else**, alert "This is my second condition".

The browser will print out the second condition, as 7 is not greater than 10.



There is also another way to do this check using the ? operator: a > b ? alert(a) : alert(b).

else if

You can use the **else if statement** to specify a new condition if the first condition is false.

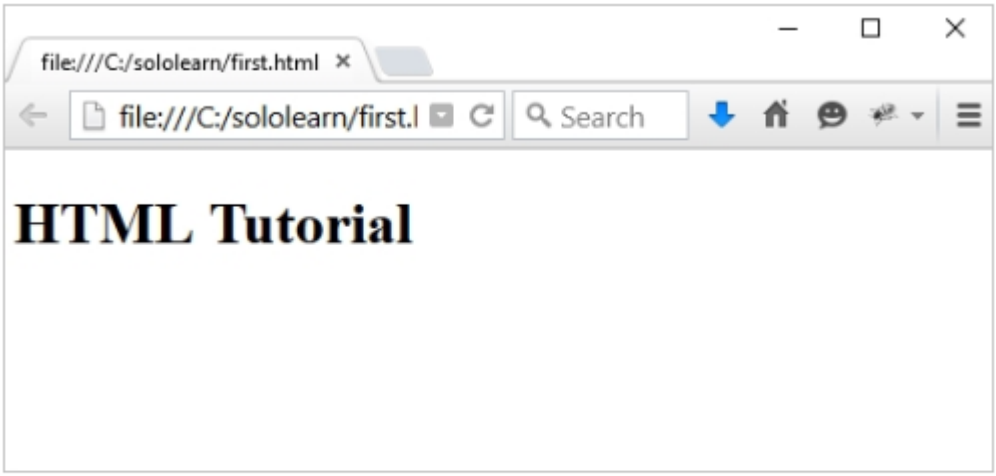
Example:

```
var course = 1;
if (course == 1) {
  document.write("<h1>HTML Tutorial</h1>");
} else if (course == 2) {
  document.write("<h1>CSS Tutorial</h1>");
} else {
  document.write("<h1>JavaScript Tutorial</h1>");
}
```

Try It Yourself

The above code says:
- **if** course is equal to 1, output "HTML Tutorial";
- **else, if** course is equal to 2, output "CSS Tutorial";
- if none of the above condition is true, then output "JavaScript Tutorial";

course is equal to 1, so we get the following result:



The final **else** statement "ends" the else if statement and should be always written after the **if** and **else if** statements.

else if

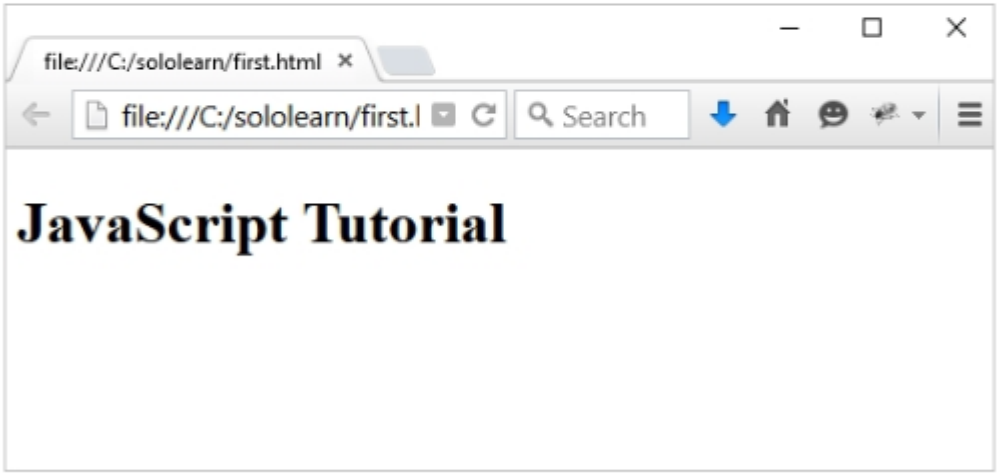
The final **else** block will be executed when **none** of the conditions is true.

Let's change the value of the **course** variable in our previous example.

```
var course = 3;
if (course == 1) {
  document.write("<h1>HTML Tutorial</h1>");
} else if (course == 2) {
  document.write("<h1>CSS Tutorial</h1>");
} else {
  document.write("<h1>JavaScript Tutorial</h1>");
}
```

Try It Yourself

The result:



You can write as many **else if** statements as you need.

Switch

In cases when you need to test for multiple conditions, writing **if else** statements for each condition might not be the best solution.
The **switch statement** is used to perform different actions based on different conditions.

Syntax:

```
switch (expression) {
  case n1:
    statements
    break;
  case n2:
    statements
    break;
  default:
    statements
}
```

The switch expression is evaluated once. The value of the expression is compared with the values of each **case**. If there is a match, the associated block of code is executed.

You can achieve the same result with multiple if...else statements, but the switch statement is more effective in such situations.

The switch Statement

Consider the following example.

```
var day = 2;
switch (day) {
 case 1:
  document.write("Monday");
  break;
 case 2:
  document.write("Tuesday");
  break;
 case 3:
  document.write("Wednesday");
  break;
 default:
  document.write("Another day");
}

// Outputs "Tuesday"
```

Try It Yourself

You can have as many **case** statements as needed.

The break Keyword

When JavaScript code reaches a **break** keyword, it breaks out of the switch block.
This will stop the execution of more code and case testing inside the block.

Usually, a **break** should be put in each case statement.

The default Keyword

The **default** keyword specifies the code to run if there is no case match.

```
var color ="yellow";
switch(color) {
  case "blue":
    document.write("This is blue.");
    break;
  case "red":
    document.write("This is red.");
    break;
  case "green":
    document.write("This is green.");
    break;
  case "orange":
    document.write("This is orange.");
    break;
  default:
    document.write("Color not found.");
}

//Outputs "Color not found."
```

Try It Yourself

> The default block can be omitted, if there is no need to handle the case when no match is found.

Loops

Loops can execute a block of code a number of times. They are handy in cases in which you want to run the same code repeatedly, adding a different value each time.

JavaScript has three types of loops: **for**, **while**, and **do while**.

The **for** loop is commonly used when creating a loop.

The syntax:

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

Statement 1 is executed before the loop (the code block) starts.
Statement 2 defines the condition for running the loop (the code block).
Statement 3 is executed each time after the loop (the code block) has been executed.

> As you can see, the **classic for loop** has **three** components, or statements.

The For Loop

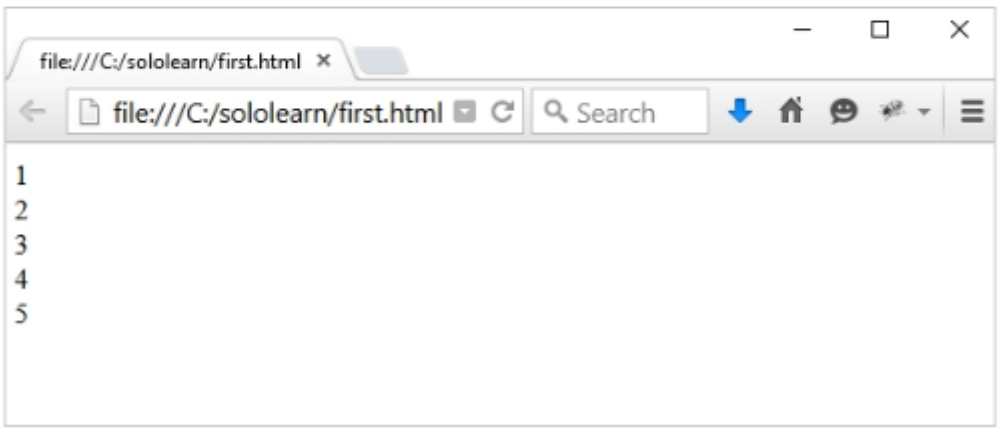
The example below creates a **for** loop that prints numbers 1 through 5.

```
for (i=1; i<=5; i++) {
  document.write(i + "<br />");
}
```

Try It Yourself

In this example, **Statement 1** sets a variable before the loop starts (var i = 1).
Statement 2 defines the condition for the loop to run (i must be less than or equal to 5).
Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Result:



Statement 1 is optional, and can be omitted, if your values are set before the loop starts.

```
var i = 1;
for (; i<=5; i++) {
  document.write(i + "<br />");
}
```

Try It Yourself

Also, you can initiate more than one value in **statement 1**, using **commas** to separate them.

```
for (i=1, text=""; i<=5; i++) {
  text = i;
  document.write(i + "<br />");
}
```

Try It Yourself

ES6 introduces other for loop types; you can learn about them in the ES6 course later.

The For Loop

If **statement 2** returns true, the loop will start over again, if it returns false, the loop will end.
Statement 2 is also optional.

If you omit statement 2, you must provide a **break** inside the loop. Otherwise, the loop will never end.

Statement 3 is used to change the initial variable. It can do anything, including negative increment (i--), positive increment (i = i + 15), or anything else.
Statement 3 is also optional, and it can be omitted if you increment your values inside the loop.

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

Try It Yourself

You can have multiple nested for loops.

The While Loop

The **while** loop repeats through a block of code, as long as a specified condition is **true**.

Syntax:

```
while (condition) {
  code block
}
```

The **condition** can be any conditional statement that returns true or false.

The While Loop

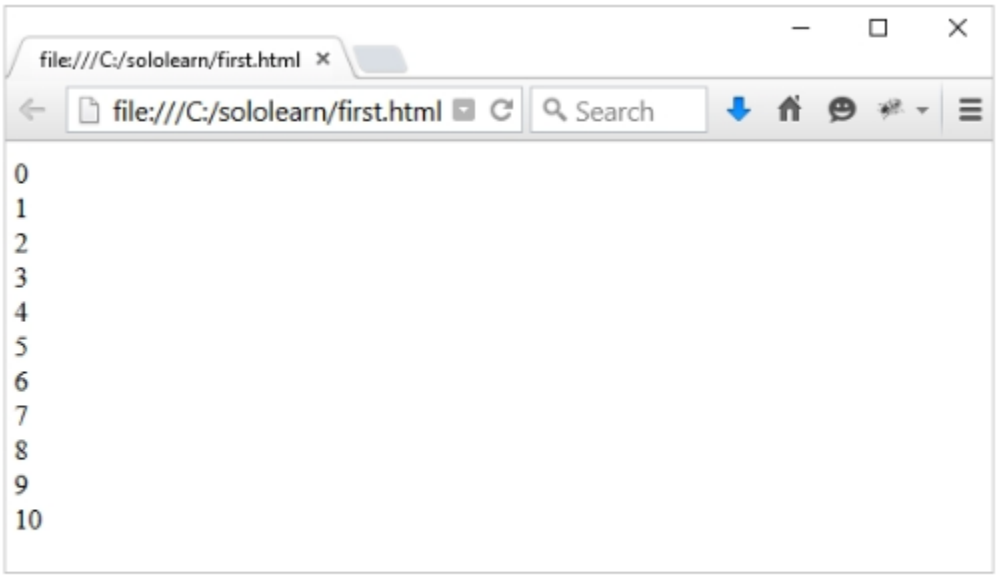
Consider the following example.

```
var i=0;
while (i<=10) {
  document.write(i + "<br />");
  i++;
}
```

Try It Yourself

The loop will continue to run as long as i is less than, or equal to, 10. Each time the loop runs, it will increase by 1.

This will output the values from 0 to 10.



Be careful writing conditions. If a condition is always true, the loop will run forever.

The While Loop

If you forget to increase the variable used in the condition, the loop will never end.

Make sure that the condition in a while loop eventually becomes **false**.

The Do...While Loop

The **do...while** loop is a variant of the while loop. This loop will execute the code block once, **before** checking if the condition is true, and then it will repeat the loop as long as the condition is true.

Syntax:

```
do {
  code block
}
while (condition);
```

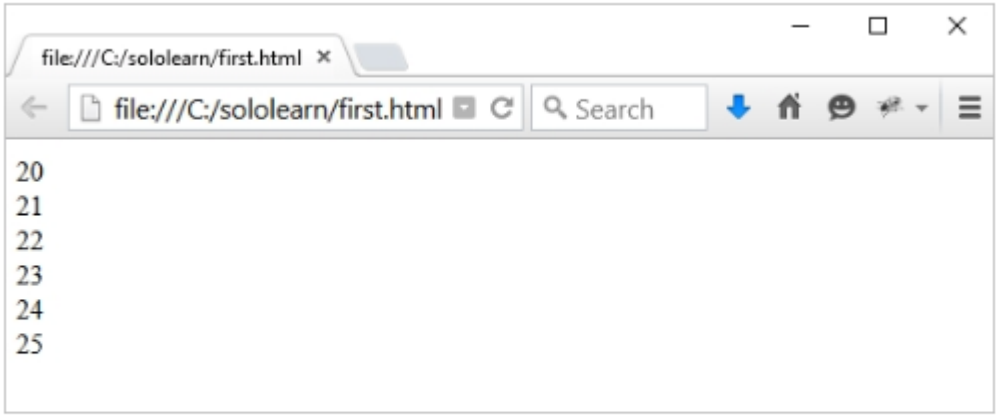
Note the **semicolon** used at the end of the do...while loop.

Example:

```
var i=20;
do {
 document.write(i + "<br />");
 i++;
}
while (i<=25);
```

Try It Yourself

This prints out numbers from 20 to 25.



The loop will always be executed **at least once**, even if the condition is false, because the code block is executed before the condition is tested.

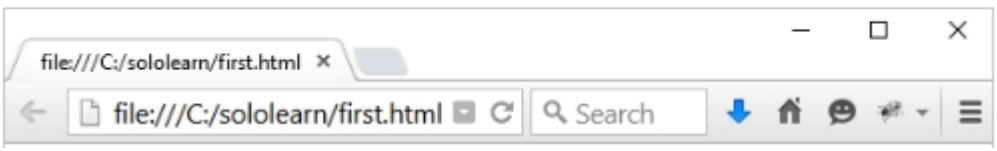
Break

The **break** statement "jumps out" of a loop and continues executing the code after the loop.

```
for (i = 0; i <= 10; i++) {
  if (i == 5) {
    break;
  }
  document.write(i + "<br />");
}
```

Try It Yourself

Once i reaches 5, it will break out of the loop.



0
1
2
3
4

You can use the return keyword to return some value immediately from the loop inside of a function. This will also break the loop.

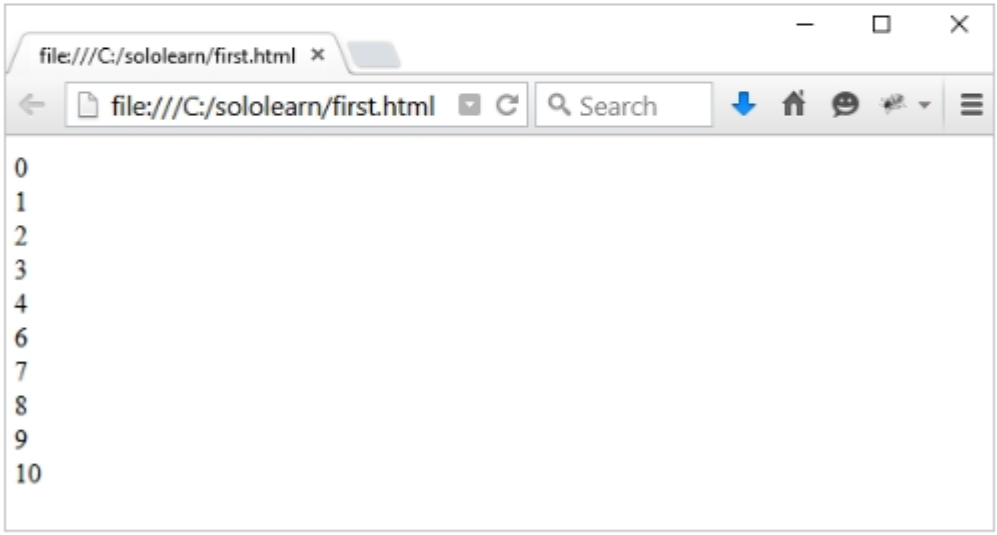
Continue

The **continue** statement breaks only one iteration in the loop, and continues with the next iteration.

```
for (i = 0; i <= 10; i++) {
  if (i == 5) {
    continue;
  }
  document.write(i + "<br />");
}
```

Try It Yourself

Result:



The value 5 is not printed, because **continue** skips that iteration of the loop.

End.