

# Working in binary

## Introduction

It is common knowledge that computers think in zeroes and ones. But, how this works in practice is a little more complex than one might initially think. You previously explored how computers utilize binary data to store and process information. In this reading, you will learn more about binary, including how to work with Boolean logic, truth tables and gates.

You may think that using only two inputs is restrictive. However, it is extraordinarily versatile and offers a broad range of options if combined with Boolean algebra and circuits.

## Boolean logic

A Boolean function maps two inputs to a value. These inputs are limited to two states. These states can be considered: **on/off**, **true/false** or **1/0**. To define a Boolean function, you only need to specify the output from its inclusion. Here is an example of 4 functions:

$\text{NOT}(x) = \begin{cases} 1 & \text{if } x \text{ is } 0 \\ 0 & \text{if } x \text{ is } 1 \end{cases}$	$\text{OR}(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ (or both) is } 1 \\ 0 & \text{otherwise} \end{cases}$
$\text{AND}(x,y) = \begin{cases} 1 & \text{if both } x \text{ and } y \text{ are } 1 \\ 0 & \text{otherwise} \end{cases}$	$\text{XOR}(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are different} \\ 0 & \text{otherwise} \end{cases}$

**NOT** takes in one value **x** and resolves it to either a **0** or **1**. **OR** takes two arguments (**x**, **y**) to generate an output of **1**. Only one of the **x** or **y** values must be **1**. If both values are **0**, the output is **0**. **AND** requires both inputs to be **1** before it can generate a **1**. Finally, **XOR** will take any two values and determine if they are the same; if so, the output is **1**. In all other cases, the result is **0**.

NOT	! x
AND	x && y

OR  
XOR

$x \mid y$   
 $x \wedge y$

The same concepts can be represented computationally. In this table, the Boolean expressions are on the left, and their computational symbols are on the right. Most commonly, they are combined with conditional statements or iterators. So, you could expect code that resembles the following code snippet:

```
Boolean x = false;
do
{
    System.out.println("executed repeatedly");
}
while(!x);
```

In the above code, a Boolean has been set to false. A **do/while** loop then continually executes the code found within the do until the value for **x** switches to **true**. Generally, you would be looking for a specific outcome, and using Boolean logic enables you to keep looking until the result becomes **true**. Notice how the **NOT** function is applied in the **while** loop to test if another iteration should exist.

## Truth tables

Given the finite number of outputs to the Boolean functions, it is possible to plot all permutations to a table.

NOT	
x	x'
0	1
1	0

OR		
x	y	x+y
0	0	1
0	1	1
1	0	1
1	1	1

AND		
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1





XOR		
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

This image demonstrates all the permutations for each of the Boolean functions. The **X** and **Y** columns denote whether

it receives a **0** or **1**, and the **XY** indicates the eventual outcome. The **NOT** function only has one output and input, so the table is only **2 x 2**. In comparison, the other three examples all take two inputs and generate a single result.

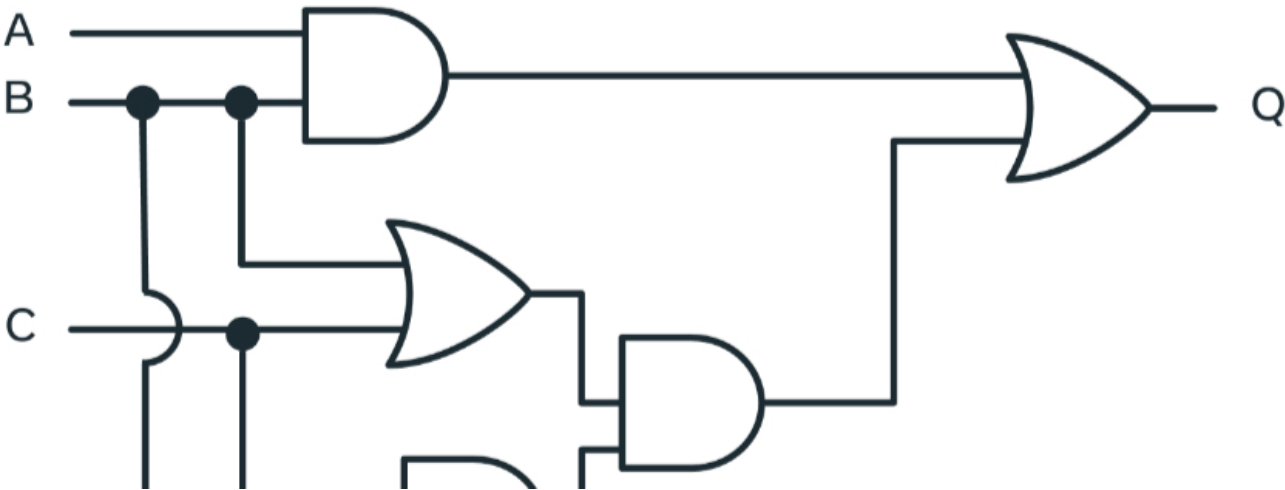
Gates

The fundamental building block for digital logic circuits is the gate. The logical functions are then implemented through their interconnectedness. A gate is an electronic circuit that generates a Boolean output from its inputs.

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

In the above image, the four basic Boolean functions are illustrated. Graphical Symbol is how these gates are represented on diagrams. You will often encounter circuit diagrams that host a series of interconnected gates. The Algebraic function demonstrates how these gates are described when written as formulae.

Finally, the truth tables outline the outcomes from a given input to the gate. On a primitive level, each input denoted by **A** and **B** represents a current that can be passed through the circuit. These inputs may be connected to a switch, or a button, that can be activated causing a **0** or **1** to be transmitted. The expression of this Boolean logic builds when circuit gates are combined to form complex circuits. The final output **Q** is determined from the machinations performed on the inputs **A**, **B**, and **C**.





## Conclusion

This reading demonstrates how a simple **0** and **1** can be combined and amplified through Boolean logic, truth tables and logic gates to generate more complex outputs. You have learned more about binary, including how to work with boolean logic, truth tables and gates.