Table of Contents

Introduction

A big part of programming is asking questions, and making decisions based on the answers. If the username and password is correct – log them in. If one or the other is not correct, don't log in! If the list of all users is already full, disallow adding any additional ones. In order for our programs to behave intelligently, or do more than a single task, we need to use **logical operators** to create **boolean expressions** which will result in **true** or **false** – yes, or no.

Relational operators

Relational operators are the ways we can compare two items: two numbers, two strings, two variables, etc. We can compare more than two items if we utilize the **and** & **or** operators as well.

Equal to	Not equal to	Greater than	Less than	Greater than OR equal to	Less than OR equal to
==	!=	>	<	>=	<=

These should all be familiar from math (except in Java programming we use two equal signs == to mean “equal to”, and a != to mean “not equal to”.)

We can ask whether one number is bigger than another, but we can also check to see if textA is equal to textB, or if isProgramDone is equal to true.

Examples

These examples work for C++, C#, and Java.

Equal to

```
if ( a == b ) ...
if ( a == 5 ) ...
if ( a == 6 / 2 ) ...
if ( a == true ) ...
if ( a == false ) ...
```

Not equal to

```
if ( a != b ) ...
if ( a != 5 ) ...
if ( a != 6 / 2 ) ...
if ( a != true ) ...
if ( a != false ) ...
```

Greater than

```
if ( a > b ) ...
if ( a > 20 ) ...
if ( 5/2 > 6*3 ) ...
```

Less than

```
if ( a < b ) ...
if ( a < 100 ) ...
if ( a*2 < 50 ) ...
```

Greater than or equal to

```
if ( a >= b ) ...
if ( a >= 20 ) ...
if ( 5/2 >= 6*3 ) ...
```

Less than or equal to

```
if ( a <= b ) ...
if ( a <= 100 ) ...
if ( a*2 <= 50 ) ...
```

Logic Operators

In math, you can write an expression like:

$$0 < x < 10$$

We can do the same thing in programming, except that this inequality cannot be written all as one expression – we have to combine two expressions:

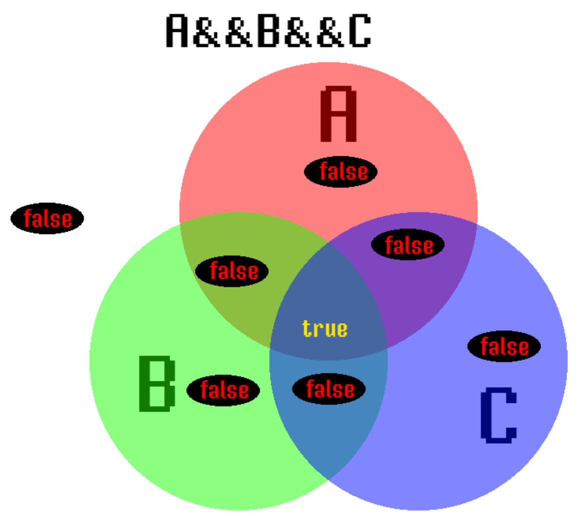
```
0 < x && x < 10
```

In programming, we can combine our **boolean expressions** with **and &&** and **or** ||, or we can negate an expression with the **not !** Operators.

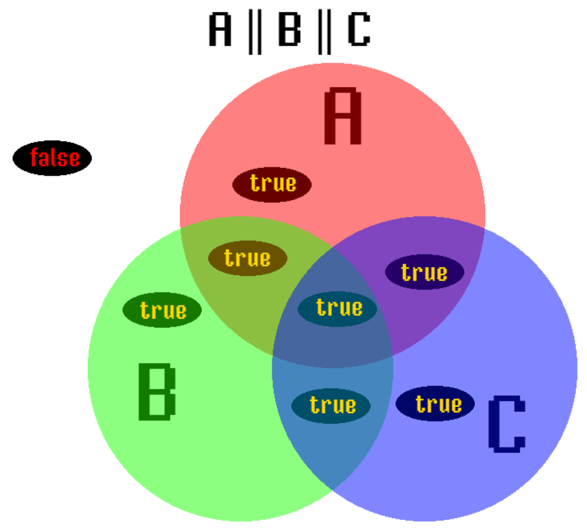
We can write separate sub-expressions and combine them with **and** or **or**, and the result of the overall-expression will depend on each one of its sub-expressions.

With the **AND operator**, **all sub-expressions must be true**. If even one of them are false, the entire expression is false.

With the **OR operator**, at least one sub-expression must be true. The rest can be false, or the rest can also be true – doesn't matter. At least one sub-expression must be true, and the entire expression is true. The entire expression is only false if **every sub-expression is false**.



A and B and C



A or B or C

Truth Tables

We can explore these relationships in-depth with something called **Truth Tables**. We use these tables to diagram the outcome of an expression based on the sub-expressions.

If we start with just one expression, “a”, it can either be True or False. Without any && or ||, the result is clear:

Sub-Expression	Total Expression
A	**A**
True	True
False	False

We can add another value and compare A and B. The left side of the truth table will be different **states** that A and B can be in – Both true, both false, or one true and one false. Then we can check the result of A && B, or A || B.

Sub-Expression		Total Expression
A	**B**	**A && B**
True	True	True
True	False	False
False	True	False
False	False	False

Sub-Expression		Total Expression
A	**B**	**A || B**
True	True	True
True	False	True
False	True	True
False	False	False

We can also build more complex expressions and use a truth table to test these out.

Sub-Expression | Total Expression

A	**B**	**C**	**A && B**	**(A && B) || C**
True	True	True	True	True
True	True	False	True	True
True	False	True	False	True
True	False	False	False	False
False	True	True	False	True
False	True	False	False	False
False	False	True	False	True
False	False	False	False	False

Mainly, keep in mind the truth table for A && B and A || B. These are important and will help you when you're building basic expressions.

For these expressions, “A” and “B” are placeholders: they could be an expression like

A:	`x < 5`	**B:**	`x > 2`
A && B:	`x < 5 && x > 2`	**A || B:**	`x < 5 || x > 2`