codecademy

Learn C#: Logic and Conditionals

Boolean Expressions

A *boolean expression* is any expression that evaluates to, or returns, a boolean value.

```
// These expressions all evaluate to a boolean value.
// Therefore their values can be stored in boolean variables.
bool a = (2 > 1);
bool b = a && true;
bool c = !false || (7 < 8);
```

Boolean Type

The `bool` data type can be either `true` or `false` and is based on the concept that the validity of all logical statements must be either true or false.
Booleans encode the science of logic into computers, allowing for logical reasoning in programs. In a broad sense, the computer can encode the truthfulness or falseness of certain statements, and based on that information, completely alter the behavior of the program.

```
bool skyIsBlue = true;
bool penguinsCanFly = false;
Console.WriteLine($"True or false, is the sky blue? {skyIsBlue}.");
// This simple program illustrates how booleans are declared. However, the real power of booleans requires additional programming constructs such as conditionals.
```

Logical Operators

Logical operators receive boolean expressions as input and return a boolean value.

The `&&` operator takes two boolean expressions and returns `true` only if they both evaluate to `true`.

The `||` operator takes two boolean expressions and returns `true` if either one evaluates to `true`.

The `!` operator takes one boolean expression and returns the opposite value.

```
// These variables equal true.
bool a = true && true;
bool b = false || true;
bool c = !false;

// These variables equal false.
bool d = true && false;
bool e = false || false;
bool f = !true;
```

Comparison Operators

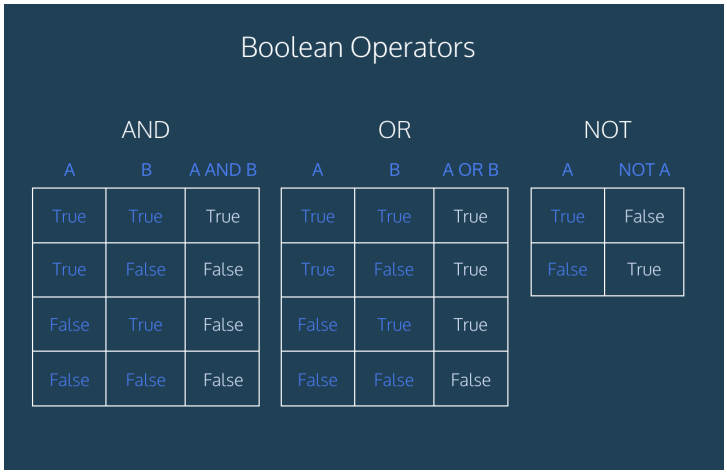
A *comparison operator*, as the name implies, compares two expressions and returns either `true` or `false` depending on the result of the comparison. For example, if we compared two `int` values, we could test to see if one number is greater than the other, or if both numbers are equal. Similarly, we can test one `string` for equality against another `string`.

```
int x = 5;
Console.WriteLine(x < 6); // Prints "True"
because 5 is less than 6.
Console.WriteLine(x > 8); // Prints "False"
because 5 is not greater than 8.

string foo = "foo";
Console.WriteLine(foo == "bar"); // Prints
"False" because "foo" does not equal "bar".
```

Truth Tables

A *truth table* is a way to visualize boolean logic. Since booleans only have two possible values, that means that we can compactly list out in a table all the possible input and output pairs for unary and binary boolean operators. The image below gives the *truth tables* for the *AND*, *OR*, and *NOT* operators. For each row, the last column represents the output given that the other columns were fed as input to the corresponding operator.



Conditional Control

Conditional statements or *conditional* control structures allow a program to have different behaviors depending on certain *conditions* being met.
Intuitively, this mimics the way humans make simple decisions and act upon them. For example, reasoning about whether to go outside might look like:

- Condition: *Is it raining outside?*
  - If it is raining outside, then *bring an umbrella.*
  - Otherwise, *do not bring an umbrella.*

We could keep adding clauses to make our reasoning more sophisticated, such as “If it is sunny, then wear sunscreen”.

Control Flow

In programming, *control flow* is the order in which statements and instructions are executed. Programmers are able to change a program's *control flow* using *control structures* such as conditionals.

Being able to alter a program's *control flow* is powerful, as it lets us adapt a running program's behavior depending on the state of the program. For example, suppose a user is using a banking application and wants to withdraw $500. We certainly want the application to behave differently depending on whether the user has $20 or $1000 in their bank account!

If Statements

In C#, an *if statement* executes a block of code based on whether or not the *boolean expression* provided in the parentheses is `true` or `false`.

If the expression is `true` then the block of code inside the braces, `{}`, is executed. Otherwise, the block is skipped over.

```
if (true) {
  // This code is executed.
  Console.WriteLine("Hello User!");
}

if (false) {
  // This code is skipped.
  Console.WriteLine("This won't be seen :(");
}
```

Break Keyword

One of the uses of the `break` keyword in C# is to exit out of `switch` / `case` blocks and resume program execution after the `switch` code block. In C#, each `case` code block inside a `switch` statement needs to be exited with the `break` keyword (or some other jump statement), otherwise the program will not compile. It should be called once all of the instructions specific to that particular `case` have been executed.

```
string color = "blue";

switch (color) {
  case "red":
    Console.WriteLine("I don't like that color.");
    break;
  case "blue":
    Console.WriteLine("I like that color.");
    break;
  default:
    Console.WriteLine("I feel ambivalent about that color.");
    break;
}
// Regardless of where the break statement is in the above switch statement,
// breaking will resume the program execution here.
Console.WriteLine("- Steve");
```

Ternary Operator

In C#, the *ternary operator* is a special syntax of the form: `condition ? expression1 : expression2`.
It takes one boolean condition and two expressions as inputs. Unlike an `if` statement, the *ternary operator* is an expression itself. It evaluates to either its first input expression or its second input expression depending on whether the condition is `true` or `false`, respectively.

```
bool isRaining = true;
// This sets umbrellaOrNot to "Umbrella" if isRaining is true,
// and "No Umbrella" if isRaining is false.
string umbrellaOrNot = isRaining ? "Umbrella" : "No Umbrella";

// "Umbrella"
Console.WriteLine(umbrellaOrNot);
```

If and Else If

A common pattern when writing multiple `if` and `else` statements is to have an `else` block that contains another nested `if` statement, which can contain another `else`, etc. A better way to express this pattern in C# is with `else if` statements. The first condition that evaluates to `true` will run its associated code block. If none are `true`, then the optional `else` block will run if it exists.

```
int x = 100, y = 80;

if (x > y)
{
  Console.WriteLine("x is greater than y");
}
else if (x < y)
{
  Console.WriteLine("x is less than y");
}
else
{
  Console.WriteLine("x is equal to y");
}
```

Switch Statements

A `switch` statement is a control flow structure that evaluates one expression and decides which code block to run by trying to match the result of the expression to each `case`. In general, a code block is executed when the value given for a `case` equals the evaluated expression, i.e, when `==` between the two values returns `true`. `switch` statements are often used to replace `if else` structures when all conditions test for equality on one value.

```
// The expression to match goes in
parentheses.
switch (fruit) {
  case "Banana":
    // If fruit == "Banana", this block will
run.
    Console.WriteLine("Peel first.");
    break;
  case "Durian":
    Console.WriteLine("Strong smell.");
    break;
  default:
    // The default block will catch
expressions that did not match any above.
    Console.WriteLine("Nothing to say.");
    break;
}

// The switch statement above is equivalent to
this:
if (fruit == "Banana") {
  Console.WriteLine("Peel first.");
} else if (fruit == "Durian") {
  Console.WriteLine("Strong smell.");
} else {
  Console.WriteLine("Nothing to say.");
}
```

Else Clause

An `else` followed by braces, `{}`, containing a code block, is called an `else` clause. `else` clauses must always be preceded by an `if` statement.

The block inside the braces will only run if the expression in the accompanying `if` condition is `false`. It is useful for writing code that runs *only if* the code inside the `if` statement is not executed.

```
if (true) {
  // This block will run.
  Console.WriteLine("Seen!");
} else {
  // This will not run.
  Console.WriteLine("Not seen!");
}

if (false) {
  // Conversely, this will not run.
  Console.WriteLine("Not seen!");
} else {
  // Instead, this will run.
  Console.WriteLine("Seen!");
}
```