In a perfect world, you would know exactly the situation in which your program will need to execute, and you could write an application as just a linear series of statements. But the world isn’t perfect, and you need the flexibility to execute certain code only when certain conditions are true. You need conditional statements.

The **if statement** executes one or more lines of code only if a given logical statement evaluates to true. If multiple lines of code are to be executed inside a single conditional check, they must be surrounded in curly braces. When the conditional lines of code are done, the flow of execution jumps back outside of the if-statement. Consider the following example:

int x = 5;

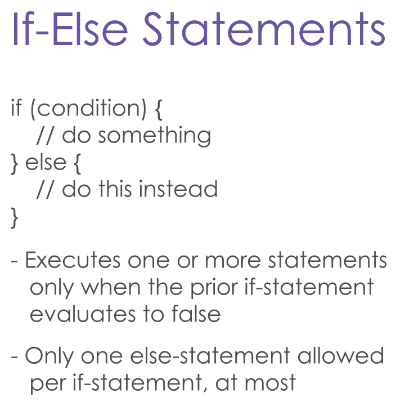
if(x == 5) {

Console.Out.WriteLine(“X is 5!”);

}

Here we have a variable, X, which is set to 5. We will check to make sure X is 5, and if it is, we will print a special message. If the check fails, no message will be printed. Running the program, we can see the message printed. If we change the value of X and re-run it, we can verify that nothing will appear.

Now what if we want to execute some code only when an if-statement evaluates to false? In other words, “If X is true, do Y, Else do Z. Well, I kind of gave it away, you can use an else statement.

Else statements must follow an if statement, and each if statement can only have one else statement.

int x = 6;

if(x == 5) {

Console.WriteLine(“X is 5!”);

}else {

Console.WriteLine(“X isn’t 5, it’s “ + x + “!”);

}

Here we have the same if statement before, testing if X is 5.

Now we’re going to add an Else statement, which will fire whenever X is not 5. When we run the program, we will see that the else statement only fires when the if condition is false.

You can also chain if-else statements – each if statement may only have zero or one else statements, but you can nest them. This will create a series of exclusive conditions; each one only being evaluated if every condition above it evaluates to “false”.

int x = 6;

if(x <= 5) {

Console.Out.WriteLine(“X is less than or equal to 5!”);

} else if (x == 6){

Console.Out.WriteLine(“X is 6!”);

}

else {

Console.Out.WriteLine(“X is neither less than or equal to 5, not is it 6.”);

}

Here we have a series of if-else statements, the first checking if X is less than or equal to five, and failing that, if it is equal to 6. This runs as expected.

But what happens when we change the chained condition to check If x is 3?

int x = 3;

if(x <= 5){

Console.WriteLine(“X is less than or equal to 5!”);

} else if (x == 6){

Console.Out.WriteLine(“X is 6!”);

}

else {

Console.Out.WriteLine(“X is neither less than or equal to 5, not is it 6.”);

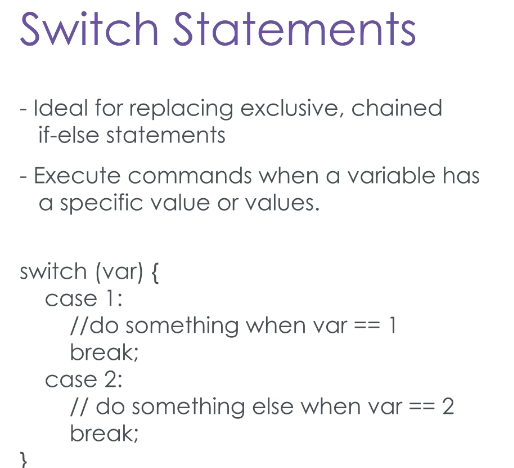
}

Now this condition will never be evaluated – if the first statement is true, if X is less than 5, the if-statement’s code will be executed and all the code in the else statement – including our nested if statement – will be skipped.If X is greater than 5, the first check will fail, but so will the second.

I’m sure you can imagine all the things you can do by chaining if-else statements. For example, you could have a menu feature, where selecting a certain option executes certain code.

**But what if that menu has 30 options?** That’s a lot of if-else statements to nest.

What if you want to have some selections to have overlapping functions? You would could end up duplicating code. Enter the switch statement.

The switch statement allows us to take a single variable of an int, bool, char, or string type, and execute code based on the values that variable holds. Let’s walk through an example.

int menu = 0;

Console.Out.WriteLine(“Please select an Option (1 or 2):”);

menu = Convert.ToInt32(Console.In.ReadLine());

switch (menu) {

case 1:

Console.Out.WriteLine(“You have selected option 1!”);

break;

case 2:

Console.Out.WriteLine(“You have selected option 2!”);

break;

default:

Console.Out.WriteLine(“Invalid selection.”);

break;

}

Here we have a switch statement, which switches on the value of the int variable “menu.”

The different switch options are called cases. Case 1 will execute if the value of menu is 1, and the code will execute until a break statement is reached. Case 2 will execute when the value of menu is 2, and so on. The default case will execute when none of the above cases are true. When we run this application a few times, we can see this in action.

Now you can finally start controlling the execution flow of your applications. Remember that with chained if-else statements, each inner if-statement will only be evaluated if all the outer statements are false.

In many cases a switch statement would be more appropriate than chained if-else’s, but each have their place.