

## Tutorial – If Statements

---

In this tutorial we are going to look at if statements. What they are, and the many different ways that we can use them.

### What are If Statements

You don't want your programs to do just one thing – you want them to do different things at different times.

In an RPG, when your character kills a monster, you want your player to get experience points to level up. You would use **if statements** to check if the player has enough experience points to level up. (For those who don't play RPG's, a character's level essentially is a way of measuring how strong the character is in this genre). Let's look at an example.

Create a new **C# console project** in Visual Studio and call it **ConditionalStatements**. Now copy the following code:

```
static void Main()
{
    int playerHealth = 100;
    int monsterDamage = 90;
    playerHealth -= monsterDamage;

    Console.WriteLine("A monster attacked you and did " + monsterDamage + "
damage. You have " + playerHealth + " health left.");

    if (playerHealth < 0)
    {
        Console.WriteLine("You died. Game over.");
    }

    Console.ReadKey();
}
```

We have variables for how much health our player has, and how much damage the monster can do. We then print out the player's health after a monster attacks.

Run the program. You will notice the text, **"You died. Game over"** didn't display. That's because the condition for our **if statement** was never **true**. If statements follow the pattern:

```
if (this condition is true)
{
    Do the code here
}
```

Now update the code:

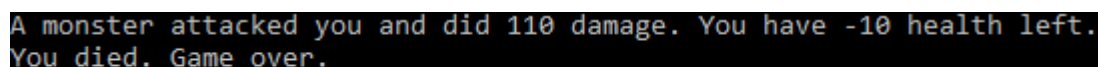
```
static void Main()
{
    int playerHealth = 100;
    int monsterDamage = 100;
    playerHealth -= monsterDamage;

    Console.WriteLine("A monster attacked you and did " + monsterDamage + "
damage. You have " + playerHealth + " health left.");

    if (playerHealth == 0)
    {
        // This only prints when health is 0
        Console.WriteLine("You died. Game over.");
    }

    Console.ReadKey();
}
```

Run the program. This time you will see the text telling you that it's game over. That's because our player's health is now 0, and our if statement says, **if playerHealth is 0**, then **print this text**.



The two == sign in our **if statement** means it's checking if the value on the left is the same as the value on the right. One = sign means you are changing the value on the left to the value on the right – it's an important difference.

The opening and closing **curly braces** define what code will run if the statement is true. If **playerHealth** is not 0, the code between the **curly braces** will never run.

## Relational Operators

You can do more than just compare if two values are the same. These different operators that compare values are called **relational operators**.

Operator	Name	Description
==	Equal	Checks if two values are the same
!=	Not Equal	Checks to see if two value <b>are not</b> the same
>	Greater Than	Checks to see if the value on the left is <b>greater</b> than the value on the right
<	Less Than	Checks to see if the value on the left is <b>less than</b> the value on the right
>=	Greater Than or Equal	Checks to see if the value on the left is <b>greater then or equal to</b> the value on the right
<=	Less Than or Equal	Checks to see if the value on the left is <b>less than or equal to</b> the value on the right

Try the following:

```
static void Main()
{
    int playerHealth = 100;
    int monsterDamage = 110;
    playerHealth -= monsterDamage;

    Console.WriteLine("A monster attacked you and did " + monsterDamage + "
damage. You have " + playerHealth + " health left.");

    if (playerHealth <= 0)
    {
        // This only prints when health is 0
        Console.WriteLine("You died. Game over.");
    }

    Console.ReadKey();
}
```

We check to see if the players health is less than **or equal** to 0. This means the player will die when his health is at 0, which we want, or if it's less than 0.

Run the program. You'll see the **playerHhealth** is -10. We can stop the players health from going below 0:

```
static void Main()
{
    int playerHealth = 100;
    int monsterDamage = 110;
    playerHealth -= monsterDamage;

    if (playerHealth < 0)
    {
        playerHealth = 0;
    }

    Console.WriteLine("A monster attacked you and did " + monsterDamage + "
damage. You have " + playerHealth + " health left.");

    if (playerHealth == 0)
    {
        // This only prints when health is 0
        Console.WriteLine("You died. Game over.");
    }

    Console.ReadKey();
}
```

Run the program. We use an **if statement** to change the players health back to 0 if it ever drops into the negatives. It is a common coding practice to make sure values don't go below or above some range that you set.

I won't demonstrate the other **relational operators** as their purpose should be fairly clear to you where you're up to. But I suggest you write out some if statements of your own and do experiments

with the other relational operators and see what happens. You're at the point where a little experimentation will help you understand better.

## Else Statements

Else statements, quite simply, are code that runs when the conditions for your **if statement** are not true. Enter the following code for an example:

```
static void Main()
{
    int playerHealth = 100;
    int monsterDamage = 70;
    playerHealth -= monsterDamage;

    if (playerHealth < 0)
    {
        playerHealth = 0;
    }

    Console.WriteLine("A monster attacked you and did " + monsterDamage + "
damage. You have " + playerHealth + " health left.");

    if (playerHealth <= 0)
    {
        // This only prints when health is 0
        Console.WriteLine("You died. Game over.");
    }
    else
    {
        Console.WriteLine("The monster prepares to attack you again!");
    }

    Console.ReadKey();
}
```

The changes we made were to cause the monster to do less damage, so that our hero would not die in one attack, and then we added an **else statement**.

Our **if statement** says that if the player has 0 or less health left, a message comes up telling the player it's game over. However, if our player has more than 0 health, the **else statement** will run the code within.

## Else If Statements

An **else statement** will always run its code if the conditions for the **if statement** are not met. If the **if statement** is true, the **else statement** will never run.

**else if** statements, on the other hand, are different. Let's look at an example:

```
static void Main()
{
    int playerHealth = 100;
    int monsterDamage = 110;
    playerHealth -= monsterDamage;

    Console.WriteLine("A monster attacked you and did " + monsterDamage + "
damage. You have " + playerHealth + " health left.");

    if (playerHealth <= 0)
    {
        // This only prints when health is 0
        Console.WriteLine("You were slain. Game over.");
    }
    else if (monsterDamage > 100)
    {
        Console.WriteLine("The monster is terrifyingly strong.");
    }
    else if (playerHealth > 0)
    {
        Console.WriteLine("The monster prepares to attack you again!");
    }

    Console.ReadKey();
}
```

Run the program. What happens?

When you have several **if statements**, the program will test if **all** of their conditions are true, and run the code for any that are true.

However, when you have **else if statements** following an **if statement**, only one of those conditions will ever run, and it will always be the first **if** or **else if statement** found to be true, because code is run from top to bottom. Even if another **else if statement** is true, it won't run. If the **if statement** is true, none of the **else if statements** will run.

This is why we see the message **"You were slain. Game over."**, because the players health is less than 0. Even though the next **else if statement** is true, the monsters damage is **greater than 100**, the statement won't run because the previous **else if statement** was true.

## Nesting If Statements

You can also put one **if statement** inside another. There may be times you want to do this, though first you should consider if there's a better way. Many times there will be.

Delete all of your previous code and enter the following:

```
static void Main()
{
    int score = 0;
    int combo = 1;
    int pointValue = 100;
    bool gameStarted = false;

    if(score == 0)
    {
        gameStarted = true;

        if (gameStarted == true)
        {
            score = combo * pointValue;
            combo++;

            Console.WriteLine("You got a combo! Your new score is: " + score);
            Console.ReadKey();
        }
    }
}
```

Run the program and see what happens.

I think it should be pretty clear to you by now. We make several variables and set them with initial values. We then have one **if statement** inside another.

First we check if the **score** is **0**. If it is it means the game has started, so we set **gameStarted** to **true**.

Then if **gameStarted** is **true**, the second **if statement** will run, increasing the players score, adding 1 to the **combo** variable (++ means add 1), then printing out the players score.

## Conclusion

Try playing with **if**, **else** and **else if statements**. Experiment manipulating with variables in different ways and print out some different values and see what you can come up with. Experimenting further with these ideas will help you to get comfortable with them.