Conditionals and Loops

The if Statement

The **if** statement is a conditional statement that executes a block of code when a condition is true.
The general form of the **if** statement is:

```
if (condition)
{
   // Execute this code when condition is true
}
```

The condition can be any expression that returns true or false.
For example:

```
static void Main(string[] args)
{
  int x = 8;
  int y = 3;

  if (x > y)
  {
    Console.WriteLine("x is greater than y");
  }
}
```

Try It Yourself

The code above will evaluate the condition **x > y**. If it is true, the code inside the if block will execute.

When only one line of code is in the if block, the curly braces can be omitted.
For example:

```
if (x > y)
Console.WriteLine("x is greater than y");
```

Relational Operators

Use **relational operators** to evaluate conditions. In addition to the less than (<) and greater than (>) operators, the following operators are available:

Operator	Description	Example	
>=	Greater than or equal to	7 >= 4	True
<=	Less than or equal to	7 <= 4	False
==	Equal to	7 == 4	False
!=	Not equal to	7 != 4	True

Example:

```
if (a == b) {
  Console.WriteLine("Equal");
}
// Displays Equal if the value of a is equal to the value of b
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

The else Clause

An optional **else** clause can be specified to execute a block of code when the condition in the **if** statement evaluates to **false**.
Syntax:

```
if (condition)
{
  //statements
}
else
{
  //statements
}
```

For example:

```
int mark = 85;

if (mark < 50)
{
  Console.WriteLine("You failed.");
}
else
{
  Console.WriteLine("You passed.");
}

// Outputs "You passed."
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

Nested if Statements

You can also include, or **nest**, if statements within another if statement.
For example:

```
int mark = 100;

if (mark >= 50) {
  Console.WriteLine("You passed.");
  if (mark == 100) {
    Console.WriteLine("Perfect!");
  }
}
else {
  Console.WriteLine("You failed.");
}

/*Outputs
You passed.
Perfect!
*/
```

Try It Yourself

You can nest an unlimited number of if-else statements.
For example:

```
int age = 17;
if (age > 14) {
  if(age > 18) {
    Console.WriteLine("Adult");
  }
  else {
    Console.WriteLine("Teenager");
  }
}
else {
  if (age > 0) {
    Console.WriteLine("Child");
  }
  else {
    Console.WriteLine("Something's wrong");
  }
}
//Outputs "Teenager"
```

Try It Yourself

Remember that all **else** clauses must have corresponding **if** statements.

The if-else if Statement

The **if-else if** statement can be used to decide among three or more actions.
For example:

```
int x = 33;

if (x == 8) {
  Console.WriteLine("Value of x is 8");
}
else if (x == 18) {
  Console.WriteLine("Value of x is 18");
}
else if (x == 33) {
  Console.WriteLine("Value of x is 33");
}
else {
  Console.WriteLine("No match");
}
//Outputs "Value of x is 33"
```

Try It Yourself

Remember, that an **if** can have zero or more **else if**'s and they must come before the last **else**, which is optional.
Once an **else if** succeeds, none of the remaining **else if**'s or **else** clause will be tested.

switch

The **switch** statement provides a more elegant way to test a variable for equality against a list of values.
Each value is called a **case**, and the variable being switched on is checked for each switch case.
For example:

```
int num = 3;
switch (num)
{
 case 1:
  Console.WriteLine("one");
  break;
 case 2:
  Console.WriteLine("two");
  break;
 case 3:
  Console.WriteLine("three");
  break;
}
//Outputs "three"
```

Try It Yourself

Each **case** represents a value to be checked, followed by a colon, and the statements to get executed if that case is matched.

A **switch** statement can include any number of **cases**. However, no two case labels may contain the same constant value.
The **break;** statement that ends each **case** will be covered shortly.

The default Case

In a switch statement, the optional **default** case is executed when none of the previous cases match.
Example:

```
int age = 88;
switch (age) {
 case 16:
  Console.WriteLine("Too young");
  break;
 case 42:
  Console.WriteLine("Adult");
  break;
 case 70:
  Console.WriteLine("Senior");
  break;
 default:
  Console.WriteLine("The default case");
  break;
}
// Outputs "The default case"
```

Try It Yourself

The **default** code executes when none of the cases matches the switch expression.

The break Statement

The role of the **break** statement is to terminate the **switch** statement.
Without it, execution continues past the matching **case** statements and falls through to the next case statements, even when the case labels don't match the switch variable.
This behavior is called **fallthrough** and modern C# compilers will not compile such code. All case and default code must end with a **break** statement.

The **break** statement can also be used to break out of a loop. You will learn about loops in the coming lessons.

while

A **while** loop repeatedly executes a block of code as long as a given condition is **true**.
For example, the following code displays the numbers 1 through 5:

```
int num = 1;
while(num < 6)
{
  Console.WriteLine(num);
  num++;
}
/* Outputs
1
2
3
4
5
*/
```

Try It Yourself

The example above declares a variable equal to 1 (int num = 1). The **while** loop checks the condition (num < 6) and, if **true**, executes the statements in its body, which increment the value of **num** by one, before checking the loop condition again.

After the 5th iteration, **num** equals 6, the condition evaluates to **false**, and the loop stops running.

The **loop body** is the block of statements within curly braces.

The while Loop

The compound arithmetic operators can be used to further control the number of times a loop runs. For example:

```
int num = 1;
while(num < 6)
{
  Console.WriteLine(num);
  num+=2;
}
/* Outputs
1
3
5
*/
```

Try It Yourself

Without a statement that eventually evaluates the loop condition to **false**, the loop will continue indefinitely.

The while Loop

We can shorten the previous example, by incrementing the value of **num** right in the condition:

```
int num = 0;
while(++num < 6)
  Console.WriteLine(num);
```

Try It Yourself

What do you think, is there a difference between **while(num++ < 6)** and **while(++num < 6)**?
Yes! The loop **while(++num < 6)** will execute 5 times, because pre-increment increases the value of x before checking the num < 6 condition, while post-increment will check the condition before increasing the value of num, making **while(num++ < 6)** execute 6 times.

The for Loop

A **for** loop executes a set of statements a specific number of times, and has the syntax:

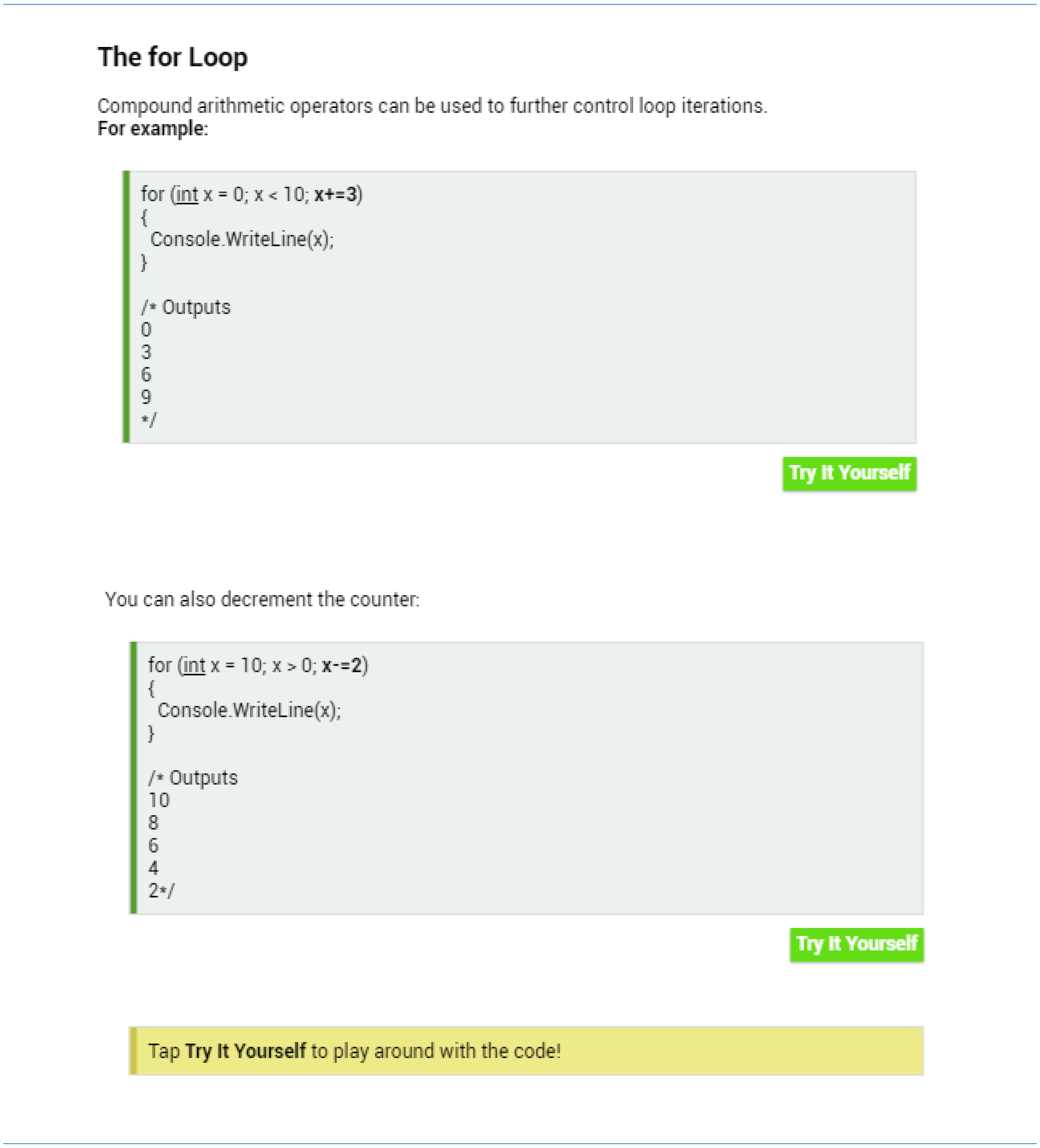
```
for ( init; condition; increment ) {
  statement(s);
}
```

A counter is declared once in **init**.
Next, the **condition** evaluates the value of the counter and the body of the loop is executed if the condition is true.
After loop execution, the **increment** statement updates the counter, also called the loop control variable.
The condition is again evaluated, and the loop body repeats, only stopping when the condition becomes **false**.
For example:

```
for (int x = 10; x < 15; x++)
{
  Console.WriteLine("Value of x: {0}", x);
}
/*
Value of x: 10
Value of x: 11
Value of x: 12
Value of x: 13
Value of x: 14
*/
```

Try It Yourself

Note the **semicolons** in the syntax.



The for Loop

Compound arithmetic operators can be used to further control loop iterations.
For example:

```
for (int x = 0; x < 10; x+=3)
{
 Console.WriteLine(x);
}

/* Outputs
0
3
6
9
*/
```

You can also decrement the counter:

```
for (int x = 10; x > 0; x-=2)
{
 Console.WriteLine(x);
}

/* Outputs
10
8
6
4
2*/
```

Tap **Try It Yourself** to play around with the code!

The for Loop

The **init** and **increment** statements may be left out, if not needed, but remember that the semicolons are mandatory.
For example, the **init** can be left out:

```
int x = 10;
for ( ; x > 0; x -= 3)
{
  Console.WriteLine(x);
}
```

Try It Yourself

You can have the increment statement in the for loop body:

```
int x = 10;
for ( ; x > 0 ; )
{
  Console.WriteLine(x);
  x -= 3;
}
```

Try It Yourself

for (; ;) {} is an infinite loop.

do-while

A **do-while** loop is similar to a **while** loop, except that a **do-while** loop is guaranteed to execute at least one time.
For example:

```
int a = 0;
do {
  Console.WriteLine(a);
  a++;
} while(a < 5);

/* Outputs
0
1
2
3
4
*/
```

Try It Yourself

Note the **semicolon** after the while statement.

do-while vs. while

If the condition of the **do-while** loop evaluates to **false**, the statements in the **do** will still run once:

```
int x = 42;
do {
  Console.WriteLine(x);
  x++;
} while(x < 10);

// Outputs 42
```

Try It Yourself

The **do-while** loop executes the statements at least once, and then tests the condition.
The **while** loop executes the statement only after testing condition.

break

We saw the use of **break** in the switch statement.
Another use of **break** is in loops: When the **break** statement is encountered inside a loop, the loop is immediately terminated and the program execution moves on to the next statement following the loop body.
For example:

```
int num = 0;
while (num < 20)
{
   if (num == 5)
     break;

   Console.WriteLine(num);
   num++;
}

/* Outputs:
0
1
2
3
4
*/
```

Try It Yourself

If you are using nested loops (i.e., one loop inside another loop), the **break** statement will stop the execution of the innermost loop and start executing the next line of code after the block.

continue

The **continue** statement is similar to the **break** statement, but instead of terminating the loop entirely, it skips the current iteration of the loop and continues with the next iteration.
For example:

```
for (int i = 0; i < 10; i++) {
 if (i == 5)
   continue;

 Console.WriteLine(i);
}
/* Outputs:
0
1
2
3
4
6
7
8
9
*/
```

Try It Yourself

As you can see, number 5 is not printed, as the **continue** statement skips the remaining statements of that iteration of the loop.

Logical Operators

Logical operators are used to join multiple expressions and return **true** or **false**.

Operator	Name of Operator	Form
&&	**AND** Operator	x && y
||	**OR** Operator	x || y
!	**NOT** Operator	! x

The **AND** operator (&&) works the following way:

Left Operand	Right Operand	Result
false	false	**false**
false	true	**false**
true	false	**false**
true	true	**true**

For example, if you wish to display text to the screen only if **age** is greater than 18 AND **money** is greater than 100:

```
int age = 42;
double money = 540;
if(age > 18 && money > 100) {
  Console.WriteLine("Welcome");
}
```

Try It Yourself

The AND operator was used to combine the two expressions.

With the AND operator, both operands must be **true** for the entire expression to be **true**.

AND

You can join more than two conditions:

```
int age = 42;
int grade = 75;
if(age > 16 && age < 80 && grade > 50)
  Console.WriteLine("Hey there");
```

Try It Yourself

The entire expression evaluates to **true** only if all of the conditions are **true**.

The OR Operator

The **OR** operator (||) returns **true** if any one of its operands is **true**.

Left Operand	Right Operand	Result
false	false	**false**
false	true	**true**
true	false	**true**
true	true	**true**

For Example:

```
int age = 18;
int score = 85;
if (age > 20 || score > 50) {
   Console.WriteLine("Welcome");
}
```

Try It Yourself

You can join any number of logical **OR** statements you want.
In addition, multiple **OR** and **AND** statements may be joined together.

Logical NOT

The logical **NOT** (!) operator works with just a single operand, reversing its logical state. Thus, if a condition is **true**, the NOT operator makes it **false**, and vice versa.

Right Operand	Result
true	**false**
false	**true**

```
int age = 8;
if ( !(age >= 16) ) {
  Console.Write("Your age is less than 16");
}

// Outputs "Your age is less than 16"
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

The ? : Operator

Consider the following example:

```
int age = 42;
string msg;
if(age >= 18)
  msg = "Welcome";
else
  msg = "Sorry";

Console.WriteLine(msg);
```

Try It Yourself

The code above checks the value of the **age** variable and displays the corresponding message to the screen.
This can be done in a more elegant and shorter way by using the **?: operator**, which has the following form:

```
Exp1 ? Exp2 : Exp3;
```

The ?: operator works the following way: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.
So, the example above can be replaced by the following:

```
int age = 42;
string msg;
msg = (age >= 18) ? "Welcome" : "Sorry";
Console.WriteLine(msg);
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

Basic Calculator

Now let's create a simple project that repeatedly asks the user to enter two values and then displays their sum, until the user enters exit.
We start with a **do-while** loop that asks the user for input and calculates the **sum**:

```
do {
  Console.Write("x = ");
  int x = Convert.ToInt32(Console.ReadLine());

  Console.Write("y = ");
  int y = Convert.ToInt32(Console.ReadLine());

  int sum = x+y;
  Console.WriteLine("Result: {0}", sum);
}
while(true);
```

This code will ask for user input infinitely. Now we need to handle the "exit".

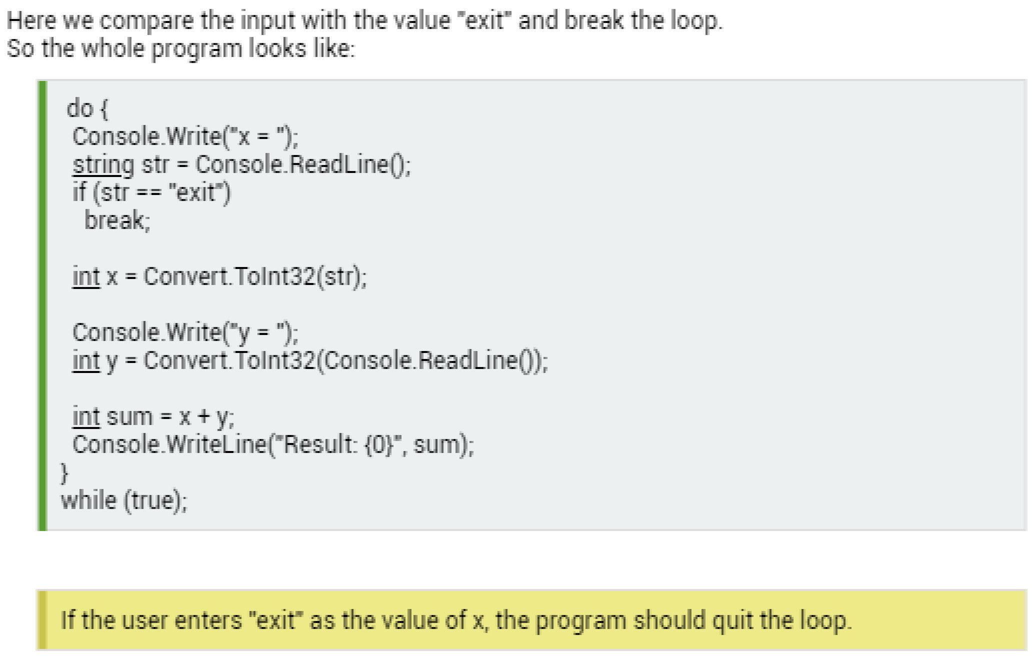
If the user enters a non-integer value, the program will crash from a conversion error. We will learn how to handle errors like that in the coming modules.

Basic Calculator

If the user enters "exit" as the value of x, the program should quit the loop. To do this, we can use a **break** statement:

```
Console.Write("x = ");
string str = Console.ReadLine();
if (str == "exit")
  break;

int x = Convert.ToInt32(str);
```


Here we compare the input with the value "exit" and break the loop.
So the whole program looks like:

```
do {
 Console.Write("x = ");
 string str = Console.ReadLine();
 if (str == "exit")
   break;

 int x = Convert.ToInt32(str);

 Console.Write("y = ");
 int y = Convert.ToInt32(Console.ReadLine());

 int sum = x + y;
 Console.WriteLine("Result: {0}", sum);
}
while (true);
```

If the user enters "exit" as the value of x, the program should quit the loop.

End.