For

Re-Iterate

Very often in programming, we encounter the situation, where we want to use a loop to iterate the same routine multiple times, e.g. 10 times, for each player:

Using goto

```
int counter = 0; // initial-statement
LoopStart:
if (counter < 10) // condition-statement
{
  Console.WriteLine("Hello, Player " + counter); // loop-body
  counter++; // iteration-statement
  goto LoopStart:
}
```

While

Using while to simplify:

```
int counter = 0; // initial-statement
while (counter < 10) // condition-statement
{
  Console.WriteLine("Hello, Player " + counter); // loop-body
  counter++; // iteration-statement
}
```

For

But even that is quite a lot of code for a task that we repeat so often. Is there a better solution?

Queue the **For-loop**:

```
int counter = 0;
// initial-statement; condition-statement; iteration-statement
for(int counter = 0; counter < 10; counter++))
{
  Console.WriteLine("Hello, Player " + counter); // loop-body
}
```

A lot less code for exactly the same thing with a **for-loop**!

Syntax

The **for-loop** consists of three main parts and the code body:

```
for(initialStatement; conditionStatement; iterationStatement)
{
  //for-loop-body
}
```

Where the **initial-statement** is executed **once** at the start of the loop.
Condition-statement is checked next **every** loop:

- It is a bool expression that needs to be true for the **for-loop** to continue
- If it returns false, then the body of the loop is not executed anymore.

Every time it loops, the **iteration-statement** is executed next and then jumps back to the **condition-statement** to check if the condition is still true.

Example...

Here’s one example that is one of the most traditional ways of using a **for-loop**:

```
for (int i  = 0; i < 5; i++) {
  Console.Write(i);
}
```

Output:

- 01234

Initial statement: **int i = 0**
Condition statement: **i < 5**
Iteration statement: **i++**
Loop body code: **Console.WriteLine($”Iteration {i}”);**

... 2

You can read a for-loop of this kind as "Repeat something 7 times":

```
// This will be executed for: 0123456
for(int i = 0; i < 7; i++) {
   // do something here
}
```

.. 3

You could start counting at 1, too, and/or use a different comparison operator <=. But this is very uncommon, we usually start counting at 0

```
// This will be executed for: 12345
for (int i  = 1; i <= 5; i++) {
  Console.WriteLine("Iteration" + i);
}
```

.. 4

You could also go backwards by using -- (the decrement-operator):

```
// This will be executed for: 43210
for (int i  =  4; i >= 0; i--) {
  Console.WriteLine("Iteration" + i);
}
```

.. 5

Also, you can have a different step in the iteration:

```
// Will be executed for: i = 02468
for (int i  = 0; i < 10; i+=2) {
  Console.WriteLine("Iteration" + i);
}
```

.. 6

You could even use completely different objects, sually not recommended, as it may confuse developers:

```
// This could be a for loop that creates a monster,
// runs for as long as the monster is alive
// and has the monster attack every iteration.

for(Monster monster = new Monster(); monster.IsAlive(); monster.Attack()){
  UpdateUI();
}
```

Lastly

You can nest for-loops, if you like to.
This can be useful for example for checking all cells in a board game like chess:

```
for (int y = 0; y < 3; y++) { // iterate bottom-to-top
  for (int x = 0; x < 3; x++) { // iterate left-to-right
    Console.WriteLine($"Coordinate at x:{x}, y:{y}");
  }
}
```

Goal - Random numbers

Print 13 different random numbers between 1 and 6.

Use a **for-loop** to iterate through the execution.

Output example:

- Here’s 13 random numbers (1-6):
- 1625462513446

Goal 2 - Power of 2’s

Start at 1, multiply the number each iteration with 2 and stop after 1024.
Print each number to the console separated by -.
Use a for-Loop.

Output:

- Here's the power of 2's:
- 1-2-4-8-16-32-64-128-256-512-1024

Goal 3 - Countdown

Start at 10 and count down until 0.
Print each number to the Console.
Use a for-Loop.

Output:

- I'm counting down!
- 10..9..8..7..6..5..4..3..2..1..0..