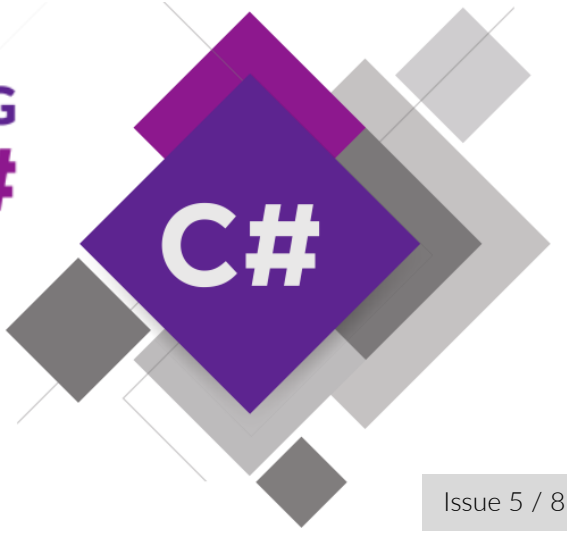
PROGRAMMING BASICS WITH C#



Issue 5 / 8

LOOPS

Learn about simple loops and how to repeat a piece of code several times, how to enter and process a sequence of numbers from the console and how to design programming logic, which requires repeating certain action several times.

Learn to use the "Turtle Graphics" and its rotate-and-move paradigm for drawing figures in C# using Visual Studio.

Dr. Svetlin Nakov

Software University – https://softuni.org

Loops

In this mini book we will get familiar how to **repeat blocks of commands**, also known in software development as "**loops**". We will write a number of **simple loops** using the **`for`** operator in its simplest form (**`for i = 1 … n`**). Finally, we will solve series of practical problems that require repeating of series of actions, using loops.

Video: Overview

Watch a video lesson to review what will we learn about loops in programming: https://youtu.be/GIE2smfXg2g.

Introduction to Simple Loops by Examples

In programming we can **execute a block of code multiple times** using a simple **for-loop** like this:

```
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine(i);
}
```

Run the above code example: https://repl.it/@nakov/for-loop-1-to-5-csharp.

The above code prints the numbers 1, 2, ..., 5. The output is as follows:

```
1
2
3
4
5
```

We can **enter multiple numbers from the console** and process them using loops like this:

1. Read the **count** **`n`** of the numbers.
2. In a **for-loop** read and process **`n`** times one single number.

This is how the above idea may work:

```
int n = int.Parse(Console.ReadLine());
long sum = 0;
for (int i = 0; i < n; i++)
{
    int num = int.Parse(Console.ReadLine());
    sum += num;
}
Console.WriteLine("Sum = {0}", sum);
```

Run the above code example: https://repl.it/@nakov/for-loop-sum-n-numbers-csharp.

The **output** from the above example may look like this (when we enter 3 numbers: **`10`**, **`20`** and **`30`**):

```
3
10
20
30
Sum = 60
```

Let's explain in greater detail how to use **simple for loops** to repeat blocks of code multiple times in C#.

For Loops (Repeating Code Blocks)

In programming it is often required **to perform a block of commands multiple times**. In order to do that, the so-called **loops** are used. Let's examine an example of a **`for` loop** that passes sequentially through the numbers from 1 to 10 and prints them:

```
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine("i = " + i);
}
```

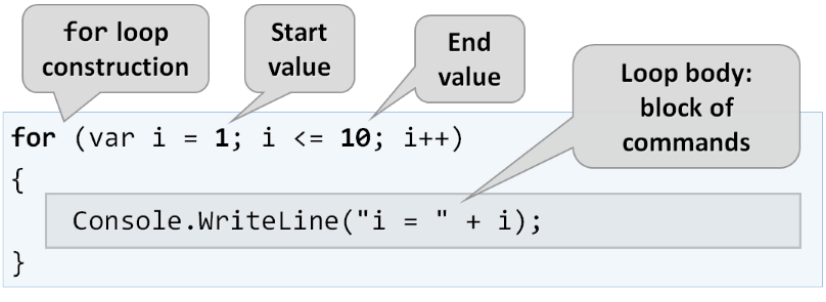
The loop starts with the **`for` operator** and passes through all values for a particular variable in a given range, for example the numbers from 1 to 10 (included), and for each value it performs a series of commands.

Video: Simple For-Loops

Watch the video about the for-loop statement: https://youtu.be/yzEamf5L1ZY.

Syntax: For-Loop

Upon declaring the loop, you can specify a **start value** and an **end value**. **The body of the loop** is usually enclosed in curly brackets **`{ }`** and represents a block of one or multiple commands. The figure below shows the structure of a **`for` loop**:



In most cases a **`for` loop** is run between **`1`** and **`n`** times (for example from 1 to 10). The purpose of the loop is to pass sequentially through the numbers 1, 2, 3, …, n and for each of them to perform a particular action. In the example above, the **`i`** variable accepts values from 1 to 10 and the current value is printed in the body of the loop. The loop repeats 10 times and each of these repetitions is called an "**iteration**".

Now, let's demonstrate how to use the for loop in practice by a few simple **examples**.

Example: Numbers from 1 to 100

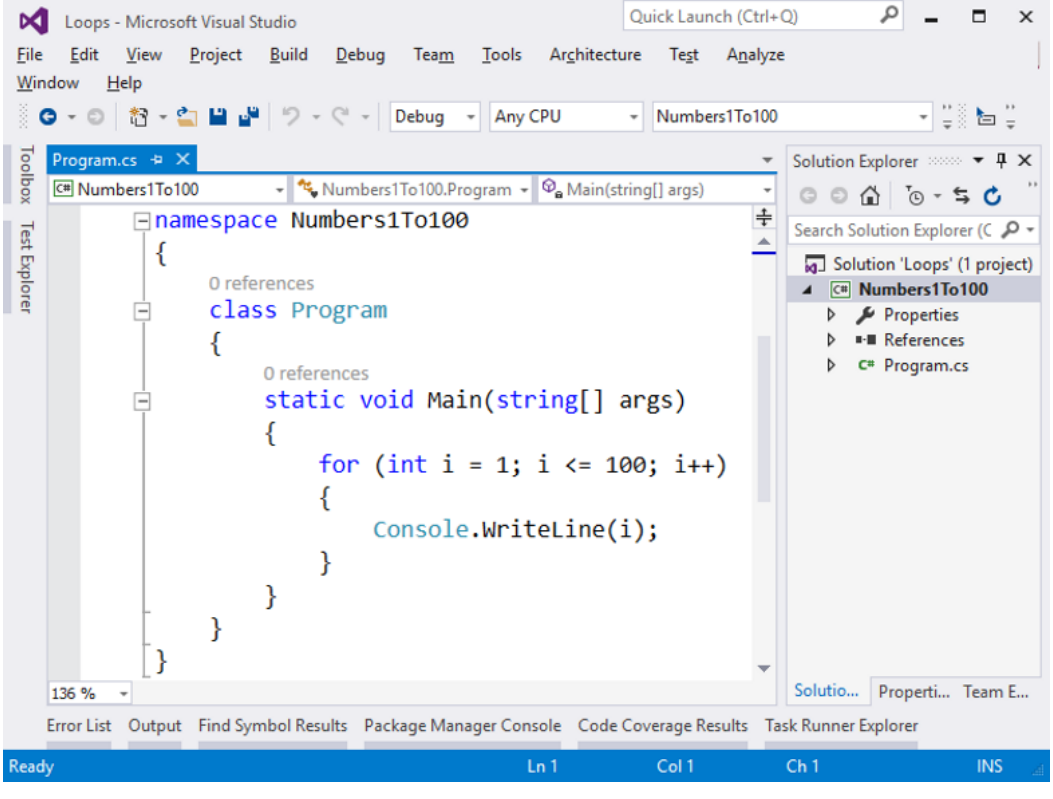
Write a program that **prints the numbers from 1 to 100**. The program does not accept input and prints the numbers from 1 to 100 sequentially, each on a separate line.

Video: Numbers 1...100

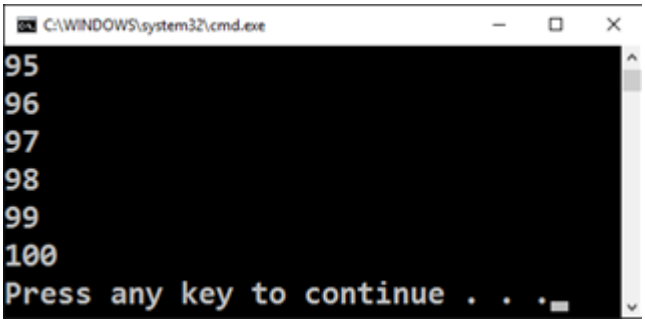
Watch the video lesson to learn how to print the numbers from 1 to 100 using a for-loop: https://youtu.be/pui8KxT_uPI.

Hints and Guidelines

We can solve the problem using a **`for` loop**, via which we will pass through the numbers from 1 to 100 using the **`i`** variable, and print the numbers in the body of the loop:



Start the program using [**Ctrl+F5**] and **test it**:



Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#0.

You should get **100 points** (fully accurate solution).

Example: Numbers up to 1000, Ending by 7

Write a program that finds all numbers within the range [**1 ... 1000**] that end in 7.

Video: Numbers 1...1000 Ending by 7

Watch the video lesson to learn how to print all numbers in the range [1...1000], ending by 7: https://youtu.be/oFJ72d5GUoo.

Hints and Guidelines

We can solve the problem by combining a **`for` loop** for passing through the numbers from 1 to 1000 and a **condition** to check if each of the numbers ends in 7. Of course, there are other solutions too, but let's solve the problem using a **loop + condition**:

```
for (int i = 1; i <= 1000; i++)
{
    if (i % 10 == 7)
    {
        // TODO: print i
    }
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#1.

Example: All Latin Letters

Write a program that prints the letters of the English Alphabet: **a, b, c, ..., z**.

Video: Latin Letters

Watch the following video lesson to learn how to print the Latin letters using a for-loop: https://youtu.be/EKNPt69wsFM.

Hints and Guidelines

It is good to know that the **`for` loops** don't only work with numbers. We can solve the task by running a **`for` loop** that passes sequentially through all letters in the English alphabet:

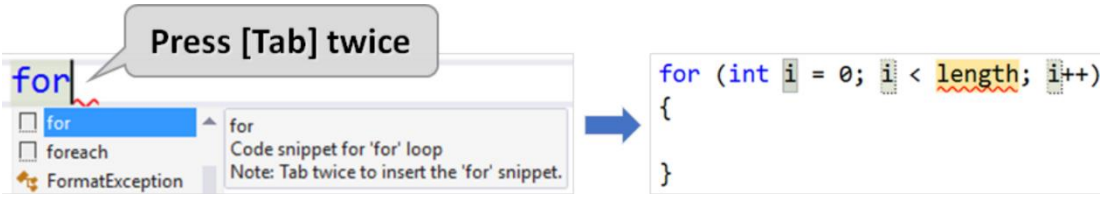
```
Console.WriteLine("Latin alphabet:");
for (int letter = 'a'; letter <= 'z'; letter++)
{
    Console.WriteLine(" " + letter);
}
Console.WriteLine();
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#2.

Code Snippet for the `for` Loop in Visual Studio

In software development, we regularly need to write loops, dozens of times a day. This is why in most development environments (IDE) there are **code snippets** for writing loops. An example of such snippet is the **snippet for `for` loop in Visual Studio**. Write **`for`** in the C# code editor in Visual Studio and **hit** [Tab] **twice**. Visual Studio will open a snippet for you and write a full **`for` loop**:



Try it yourselves, in order to master the skill of using the code snippet for **`for` loop** in Visual Studio.

Exercises: Repetitions (Loops)

Now that we are familiar with loops, it is time **to practice our newly acquired skills**, and as you know, this is achieved by a lot of code writing. Let's solve some **practical problems**.

Video: Summary

Watch this video to review what we learned: https://youtu.be/4G_oSUcx9ko.

What We Learned?

We can repeat a code block using a **`for` loop**:

```
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine("i = " + i);
}
```

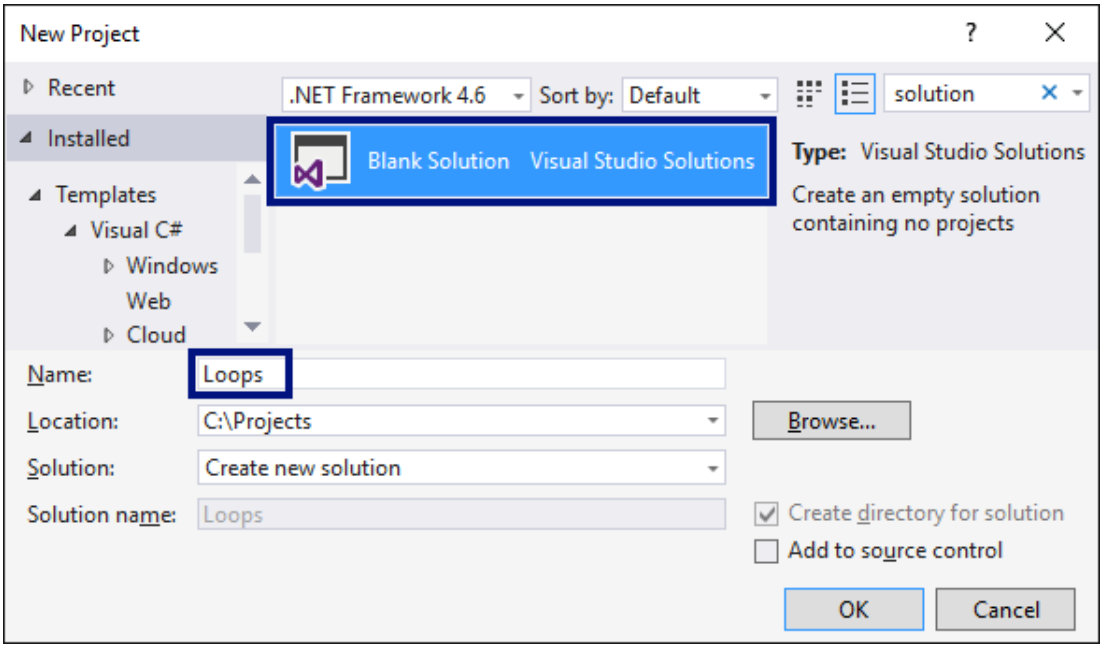
We can read a sequence of **`n`** numbers from the console this way:

```
var n = int.Parse(Console.ReadLine());
for (int i = 0; i < n; i++)
{
    var num = int.Parse(Console.ReadLine());
}
```

Blank Solution in Visual Studio

Before starting with the exercises, we will create a **Visual Studio solution** with the idea to hold the code for each exercises problem in a **separate C# project** inside the solution.

Create a **(Blank Solution)** in Visual Studio:



Set it up to **start the current project by default** (not the first one in the solution). Do that by right clicking on **Solution 'Loops'** -> [**Set StartUp Projects...**] -> [**Current selection**].

Problem: Summing up Numbers

Write a program that **inputs `n` integers and sums them up**.

- The first line of the input holds the number of integers **`n`**.
- Each of the following **`n`** lines holds an integer for summing.
- Sum up the numbers and finally print the result.

Sample Input and Output

Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
2 10 20	30	3 -10 -20 -30	-60	4 45 -20 7 11	43	1 999	999	0	0

Video: Summing Numbers

Watch this video to learn how to sum numbers in for-loop: https://youtu.be/t7PAichwl7k.

Hints and Guidelines

We can solve the problem with summing up numbers in the following way:

- We read the input number **n**.
- We initially start with a sum **sum = 0**.
- We run a loop from 1 to **n**. On each step of the loop, we read the number **num** and add it to the sum **sum**.
- Finally, we print the calculated amount **sum**.

Below is the source code for the solution:

```
Console.WriteLine("n = ");
var n = int.Parse(Console.ReadLine());
Console.WriteLine("Enter the numbers:");
var sum = 0;

for (int i = 0; i < n; i++)
{
    var num = int.Parse(Console.ReadLine());
    sum = sum + num;
}

Console.WriteLine("sum = " + sum);
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#3.

Problem: Max Number

Write a program that enters **n integers** (**n** > 0) and finds **the max number** among them (the largest number). The first line of the input, specifies the number of integers **n**. The next **n** lines hold the integers, one per line. Examples:

Sample Input and Output

Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
2 100 99	100	3 -10 20 -30	20	4 45 -20 7 99	99	1 999	999	2 -1 -2	-1

Video: Largest Number

Watch this video lesson to learn how to find the largest number among a sequence of number: https://youtu.be/KErfOdOuezE.

Hints and Guidelines

We will first enter one number **n** (the number of integers that are about to be entered). We assign the current maximum **max** an initial neutral value, for example **-10000000000000** (or **int.MinValue**). Using a **for** loop that is iterated **n-1 times**, we read one integer **num** on each iteration. If the read number **num** is higher than the current maximum **max**, we assign the value of the **num** to the **max** variable. Finally, in **max** we must have stored the highest number. We print the number on the console.

```
Console.WriteLine("n = ");
var n = int.Parse(Console.ReadLine());
var max = -1000000000000000;

for (int i = 1; i <= n; i++)
{
    var num = int.Parse(Console.ReadLine());
    if (num > max)
    {
        max = num;
    }
}

Console.WriteLine("max = " + max);
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#4.

Problem: Min Number

Write a program that inputs **n integers** (**n** > 0) and finds **the min** number among them (the smallest number). First enter the number of integers **n**, then **n** numbers additionally, one per line.

Sample Input and Output

Input	Output
2 100 99	99
3 -10 20 -30	-30
4 45 -20 7 99	-20

Video: Smallest Number

Watch this video lesson to learn how to find the smallest number among a sequence of number: https://youtu.be/IHuz-mXbhzg.

Hints and Guidelines

The problem is completely identical to the previous one, except this time we will start with another neutral starting value.

```
Console.WriteLine("n = ");
var n = int.Parse(Console.ReadLine());
var min = 1000000000000000;

for (int i = 1; i <= n; i++)
{
    var num = int.Parse(Console.ReadLine());
    if (num < min)
    {
        min = num;
    }
}

Console.WriteLine("min = " + min);
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#5.

Problem: Left and Right Sum

Write a program that converts 2 * n integers and checks whether the sum of the first n integers (left sum) equals the sum of the second n numbers (right sum). In case the sums are equal, print "Yes" + the sum, otherwise print "No" + the difference. The difference is calculated as a positive number (by absolute value). The format of the output must be identical to the one in the examples below.

Sample Input and Output

Input	Output	Input	Output
2 10 90 60 40	Yes, sum = 100	2 90 9 50 50	No, diff = 1

Video: Left and Right Sum

Watch this video lesson to learn how to find calculate the left and the right sum and their difference: https://youtu.be/s_uAuqTnC8w.

Hints and Guidelines

We will first input the number n, after that the first n numbers (left half) and sum them up. We will then proceed with inputting more n numbers (the right half) and sum them up. We calculate the difference between the sums by absolute value: **`Math.Abs(leftSum - rightSum)`**. If the difference is 0, print "Yes" + the sum, otherwise print "No" + the difference.

```
Console.WriteLine("n = ");
var n = int.Parse(Console.ReadLine());
var leftSum = 0;
var rightSum = 0;

for (int i = 0; i < n; i++)
{
    leftSum = leftSum + int.Parse(Console.ReadLine());
}
// TODO: read and calculate the rightSum

if (leftSum == rightSum)
{
    Console.WriteLine("Yes, sum = " + leftSum);
}
else
{
    var difference = Math.Abs(rightSum - leftSum);
    Console.WriteLine("No, diff = " + difference);
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#6.

Problem: Even / Odd Sum

Write a program that inputs **n integers** and checks whether **the sum of the numbers on even positions** is equal to **the sum of the numbers on odd positions**. In case the sums are equal, print **"Yes" + the sum**, otherwise, print **"No" + the difference**. The difference is calculated by absolute value. The format of the output must be identical to the examples below.

Sample Input and Output

Input	Output
4 10 50 60 20	Yes Sum = 70
4 3 5 1 -2	No Diff = 1
3 5 8 1	No Diff = 2

Video: Even / Odd Sum

Watch this video to learn how to sum the element at even and odd positions and how to calculate their equality or difference: https://youtu.be/79QsS7FI2qg.

Hints and Guidelines

We input the numbers one by one and calculate the two **sums** (of the numbers on **even** positions and the numbers on **odd** positions). Identically to the previous problem, we calculate the absolute value of the difference and print the result (**"Yes" + the sum** in case of difference of 0 or **"No" + the difference** in any other case).

```
Console.WriteLine("n = ");
var n = int.Parse(Console.ReadLine());
var oddSum = 0;
var evenSum = 0;
```

```
for (int i = 0; i < n; i++)
{
    var element = int.Parse(Console.ReadLine());
    if (i % 2 == 0)
    {
        evenSum += element;
    }
    else
    {
        oddSum += element;
    }
}
// TODO: print the sum / difference
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#7.

Problem: Sum of Vowels

Write a program that inputs **text** (string), calculates and prints **the sum of the values of vowels** according to the table below:

a	e	i	o	u
1	2	3	4	5

Sample Input and Output

Input	Output	Input	Output
hello	6 (e+o = 2+4 = 6)	bamboo	9 (a+o+o = 1+4+4 = 9)
hi	3 (i = 3)	beer	4 (e+e = 2+2 = 4)

Video: Sum of Vowels

Watch this video lesson to learn how to sum the vowels of given text: https://youtu.be/9hi9G3vRA0U.

Hints and Guidelines

We read the input text **`s`**, null the sum and run a loop from 0 to **`s.Length-1`** (text length -1). We check each letter **`s[i]`** and verify if it is a vowel, and accordingly, add its value to the sum.

```
var s = Console.ReadLine();
var sum = 0;

for (int i = 0; i < s.Length; i++)
{
    if (s[i] == 'a')
    {
        sum += 1;
    }
    else if (s[i] == 'e')
    {
        sum += 2;
    }
    else if (s[i] == 'i')
    {
        sum += 3;
    }
    else if (s[i] == 'o')
    {
        sum += 4;
    }
    else if (s[i] == 'u')
    {
        sum += 5;
    }
}

Console.WriteLine("Vowels sum = " + sum);
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#8.

Problem: Element Equal to the Sum of the Rest

Write a program that inputs **n integers** and checks whether among them there is a number equal to the sum of all the rest. If there is such an element, print **"Yes" + its value**, otherwise – **"No" + the difference between the largest element and the sum of the rest** (by absolute value).

Sample Input and Output

Input	Output	Comments
7 3 4 1 1 2 12 1	Yes Sum = 12	3 + 4 + 1 + 2 + 1 + 1 = 12
4 6 1 2 3	Yes Sum = 6	1 + 2 + 3 = 6
3 1 1 10	No Diff = 8	10 - (1 + 1) = 8
3 5 5 1	No Diff = 1	5 - (5 + 1) = 1
3 1 1 1	No Diff = 1	1 - (1 + 1) = 1

Hints and Guidelines

We must calculate the **sum** of all elements, find the **largest** of them and check the condition.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#9.

Problem: Even / Odd Positions

Write a program that reads **n numbers** and calculates **the sum**, the **min** and **max** values of the numbers on **even** and **odd** positions (counted from 1). If there is no min / max element, print **"No"**.

Sample Input and Output

Input	Output	Input	Output
6 2 3 5 4 2 1	OddSum=9, OddMin=2, OddMax=5, EvenSum=8, EvenMin=1, EvenMax=4	2 1.5 -2.5	OddSum=1.5, OddMin=1.5, OddMax=1.5, EvenSum=-2.5, EvenMin=-2.5, EvenMax=-2.5
1 1	OddSum=1, OddMin=1, OddMax=1, EvenSum=0, EvenMin=No, EvenMax=No	0	OddSum=0, OddMin=No, OddMax=No, EvenSum=0, EvenMin=No, EvenMax=No
5 3 -2 8 11 -3	OddSum=8, OddMin=-3, OddMax=8, EvenSum=9, EvenMin=-2, EvenMax=11	4 1.5 1.75 1.5 1.75	OddSum=3, OddMin=1.5, OddMax=1.5, EvenSum=3.5, EvenMin=1.75, EvenMax=1.75
1 -5	OddSum=-5, OddMin=-5, OddMax=-5, EvenSum=0, EvenMin=No, EvenMax=No	3 -1 -2 -3	OddSum=-4, OddMin=-3, OddMax=-1, EvenSum=-2, EvenMin=-2, EvenMax=-2

Hints and Guidelines

The task combines some of the previous tasks: finding the **min** and **max** value and **sum**, as well as processing of elements on **even and odd positions**. Check them out.

In the current task it is better to work with **fractions** (not integers). The sum, the min and the max value will also be fractions. We must use a **neutral starting value** upon finding the min / max value, for example **1000000000.0** or **-1000000000.0**. If the end result is the neutral value, we will print **"No"**.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#10.

Problem: Equal Pairs

There are **2 * n numbers**. The first and the second number form a **pair**, the third and the fourth number also, and so on. Each pair has a **value** – the sum of its numbers. Write a program that checks **if all pairs have equal value**.

In case the value is the same, print "Yes, value=..." + the value, otherwise, print the maximum difference between two neighboring pairs in the following format: "No, maxdiff=..." + the maximum difference.

The input consists of the number n, followed by 2*n integers, all of them one per line.

Sample Input and Output

Input	Output	Comments
3 1 2 0 3 4 -1	Yes, value=3	values = {3, 3, 3} equal values
2 1 2 2 2	No, maxdiff=1	values = {3, 4} differences = {1} max difference = 1
4 1 1 3 1 2 2 0 0	No, maxdiff=4	values = {2, 4, 4, 0} differences = {2, 0, 4} max difference = 4
1 5 5	Yes, value=10	values = {10} one value equal values
2 -1 0 0 -1	Yes, value=-1	values = {-1, -1} equal values
2 -1 2 0 -1	No, maxdiff=2	values = {1, -1} differences = {2} max difference = 2

Hints and Guidelines

We read the input numbers **in pairs**. For each pair we calculate its **sum**. While reading the input pairs, for each pair except the first one, we must calculate **the difference compared to the previous one**. In order to do that, we need to store as a separate variable the sum of the previous pair. Finally, we find the **largest difference** between two pairs. If it is **0**, print **"Yes"** + the value, otherwise – **"No"** + the difference.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/510#11.

Lab: Turtle Graphics GUI Application

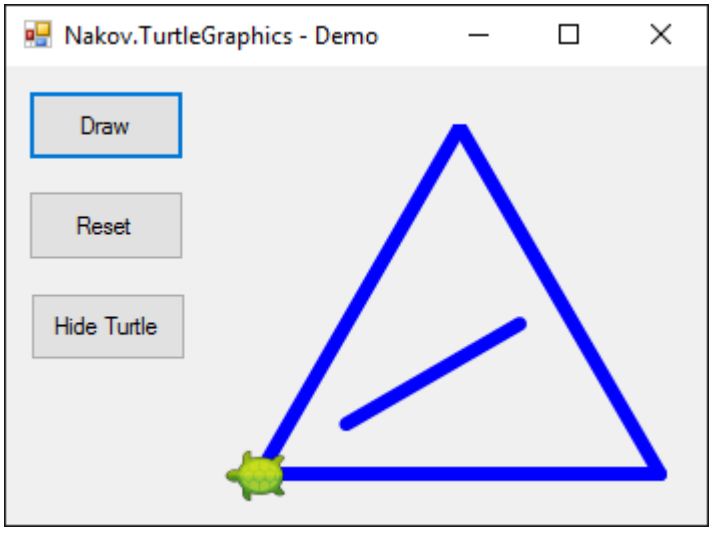
In this mini book we learned about **loops** as a programming construction that allows to repeat a particular action or a group of actions multiple times. Now let's play with them. In order to do that, we will draw some figures that will consist of a large number of repeating graphical elements, but this time we will not do it on the console, but in a graphical environment using "**turtle graphics**". It will be interesting. And it is not hard at all. Try it!

Video: Turtle Graphics

Watch the video to learn about turtle graphics and how to draw figures by moving and rotating the turtle in a Windows Forms GUI application: https://youtu.be/WwSjMHo0Fx4.

What Shall We Build?

The purpose of the following exercise is to play with a "move and rotate" **drawing library**, also known as **"turtle graphics"**. We will build a graphical application (GUI App) in which we will **draw various figures** by moving our **"turtle"** across the screen via operations like "move 100 positions ahead", "turn 30 degrees to the right", "move 50 more positions ahead". The application will look approximately like this:



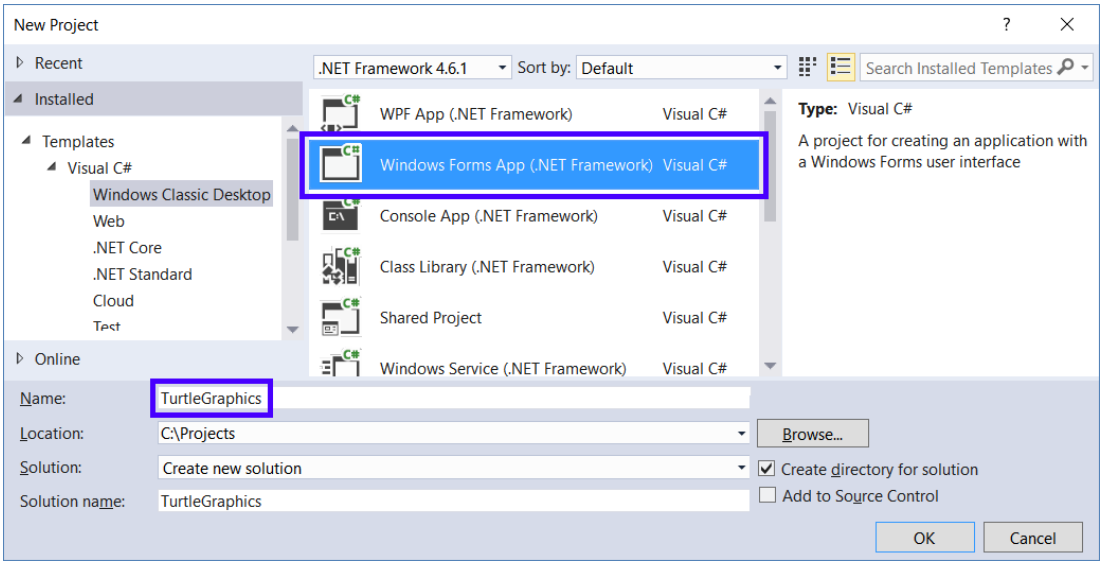
Turtle Graphics – Concepts

Let's first get familiar with **the concept of drawing "Turtle Graphics"**. Take a look at the following sources:

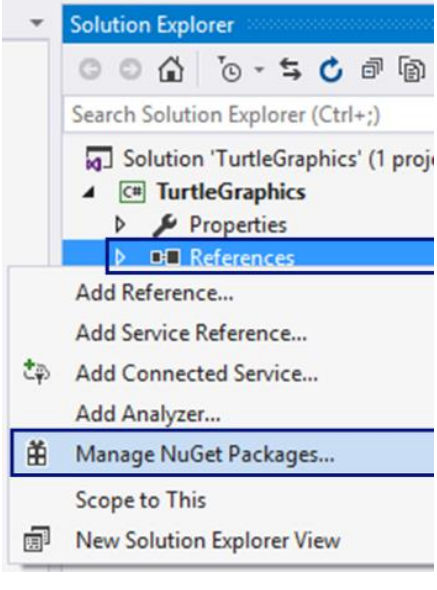
- Definition of "turtle graphics": http://c2.com/cgi/wiki?TurtleGraphics
- Article on "turtle graphics" in Wikipedia – https://en.wikipedia.org/wiki/Turtle_graphics
- Interactive online tool for drawing with a turtle – https://blockly-games.appspot.com/turtle

Creating a New C# Project

We will start by creating a new **C# Windows Forms Project**:



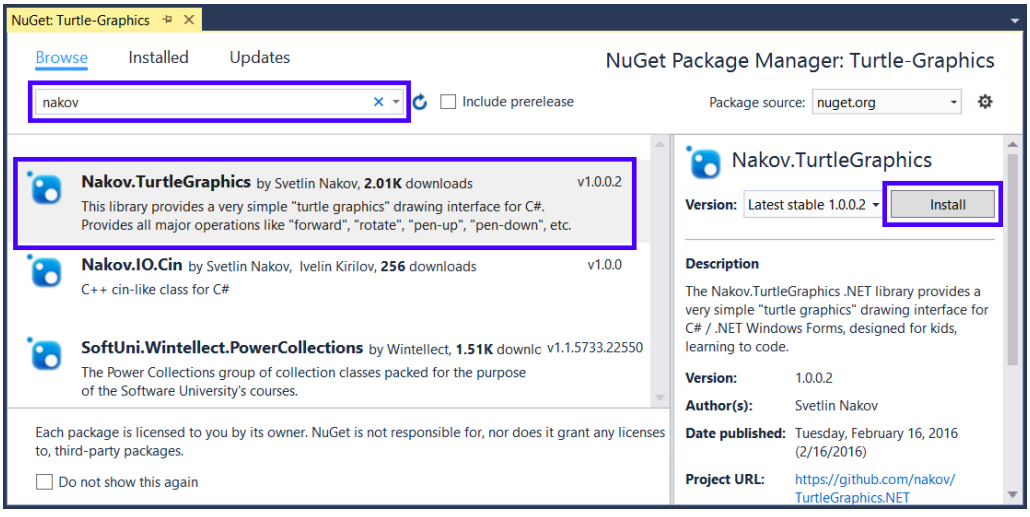
Installing Turtle Graphics NuGet Package



Install the **NuGet** package **"`Nakov.TurtleGraphics`"** to your new Windows Forms project. From Visual Studio you can add **external libraries**(packages) to an existing C# project. They add up additional functionality to our applications. The official repository for C# libraries is maintained by Microsoft and is called **NuGet** (http://www.nuget.org).

Right-click the **Solution Explorer** project and select [**Manage NuGet Packages...**].

A **NuGet** package search and installation window will open. Let's search for packages by keyword `nakov`. A few packages will be found. Select `Nakov.TurtleGraphics`. Click [**Install**] to install it to your C# project:

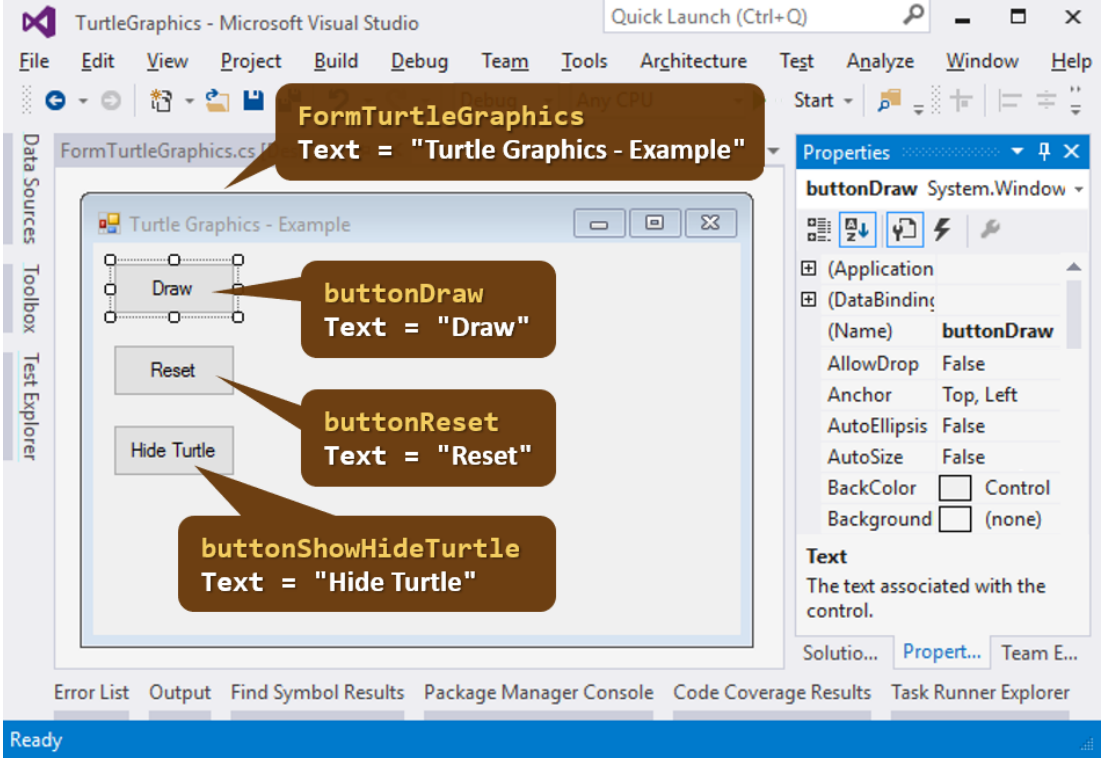


The external library `Nakov.TurtleGraphics` is already included in our C# project. It defines `Turtle` class that represents **a drawing turtle**. In order to use it, add (`Form1.cs`) in the C# code for our form. Add the following code at the top of the file:

```
using Nakov.TurtleGraphics;
```

Adding the Buttons

Now we need to add **three buttons** into the form and change their **names** and **properties** as stated below:



Implementing the [Draw] Button

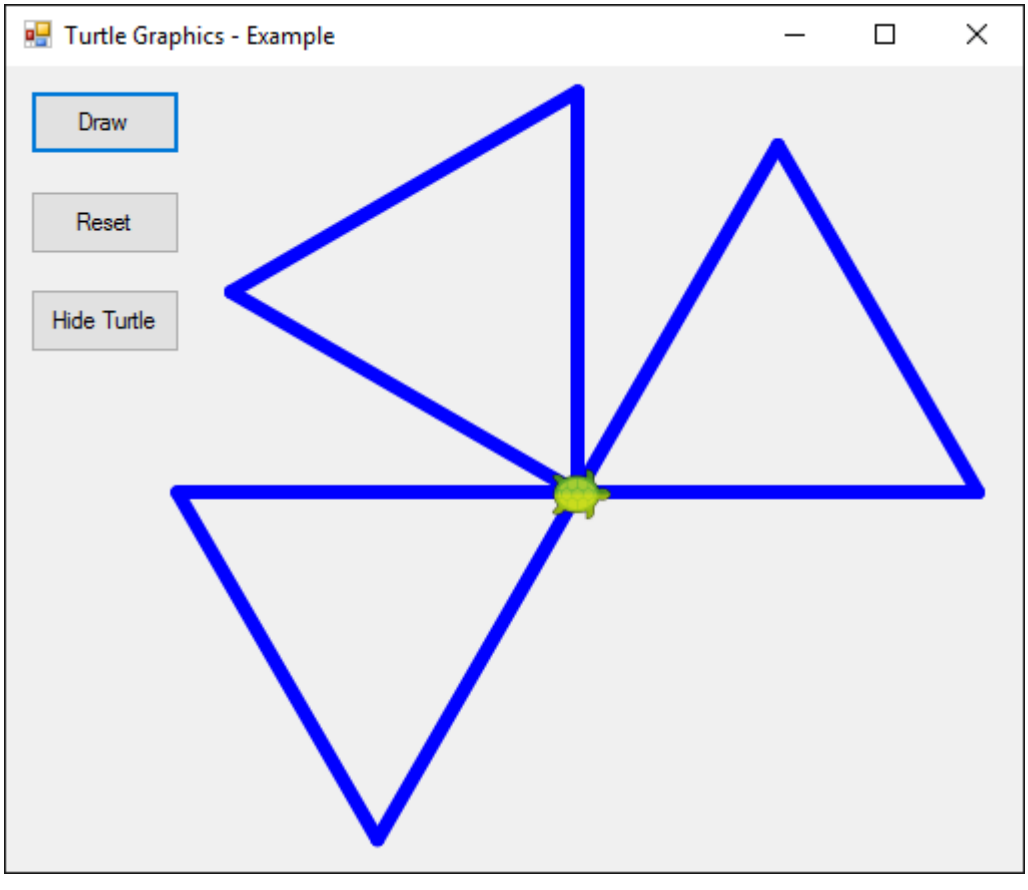
Double-click the [Draw] button in order to add the code that will be executed upon its pressing. Write the following code:

```
private void buttonDraw_Click(object sender, EventArgs e)
{
    Turtle.Rotate(30);
    Turtle.Forward(200);
    Turtle.Rotate(120);
    Turtle.Forward(200);
    Turtle.Rotate(120);
    Turtle.Forward(200);
}
```

This code moves and rotates the turtle that is initially in the center of the screen (in the middle of the form) and draws an equilateral triangle. You can edit it and play with it.

Testing the Application

Start the application by pressing [Ctrl+F5]. Test if it works (press the [Draw] button a few times):



Adding Complexity to the Turtle Drawing Code

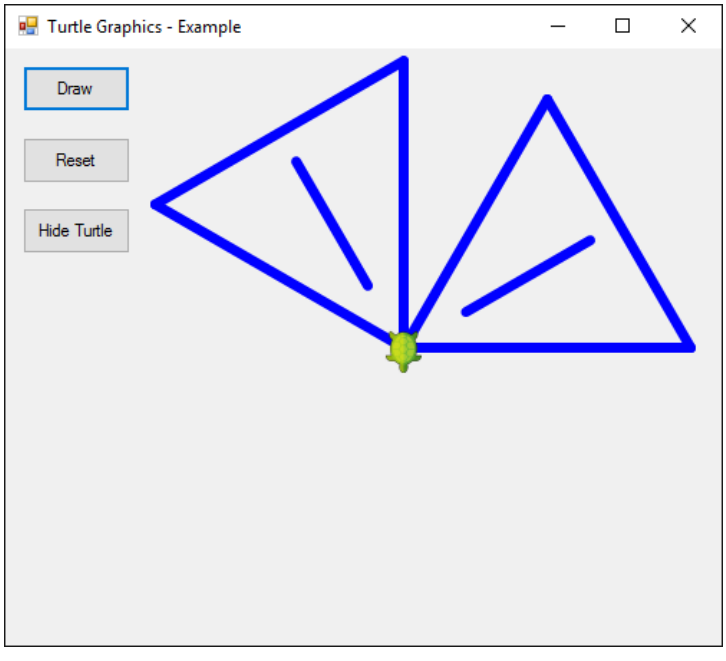
Now you can modify the turtle code and make it more complex:

```
// Assign a delay to visualize the drawing process
Turtle.Delay = 200;

// Draw a equilateral triangle
Turtle.Rotate(30);
Turtle.Forward(200);
Turtle.Rotate(120);
Turtle.Forward(200);
Turtle.Rotate(120);
Turtle.Forward(200);

// Draw a line in the triangle
Turtle.Rotate(-30);
Turtle.PenUp();
Turtle.Backward(50);
Turtle.PenDown();
Turtle.Backward(100);
Turtle.PenUp();
Turtle.Forward(150);
Turtle.PenDown();
Turtle.Rotate(30);
```

Start the application again by pressing [Ctrl+F5]. Test whether the new turtle program works:



Now our turtle draws more complex figures via a nice animated motion.

Implementing the [Reset] Button

Now let's write the code for the other two buttons. The purpose of the [Reset] button is to delete the graphics and to start drawing from the beginning:

```
private void buttonReset_Click(object sender, EventArgs e)
{
    Turtle.Reset();
}
```

Implementing the [Show / Hide Turtle] Buttons

The purpose of the [Show / Hide Turtle] button is to show or hide the turtle:

```
private void buttonShowHideTurtle_Click(object sender, EventArgs e)
{
    if (Turtle.ShowTurtle)
    {
        Turtle.ShowTurtle = false;
        this.buttonShowHideTurtle.Text = "Show Turtle";
    }
    else
    {
        Turtle.ShowTurtle = true;
        this.buttonShowHideTurtle.Text = "Hide Turtle";
    }
}
```

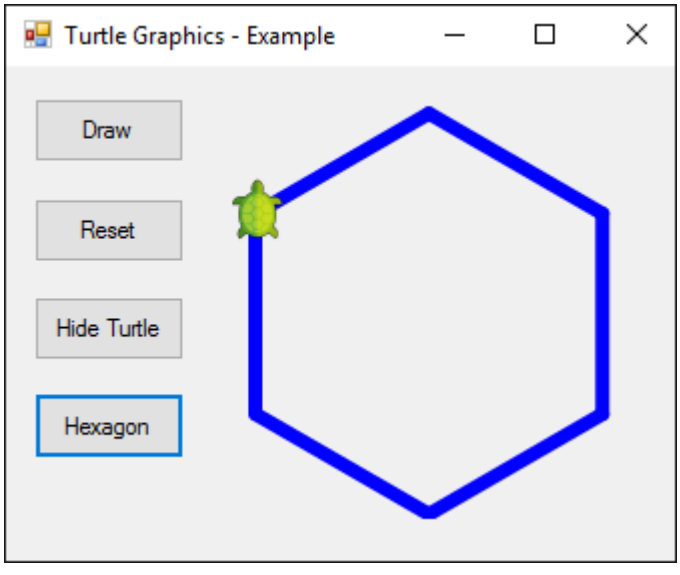
Once again, start the application by [Ctrl+F5] and test whether it works correctly.

Exercises: Turtle Graphics

Now, it is your time to draw a few figures with the turtle, using sequences of moves and rotations. Add a new additional button for drawing each of the figures, shown below.

Problem: * Draw a Hexagon with the Turtle

Add a [Hexagon] button that draws a regular hexagon:



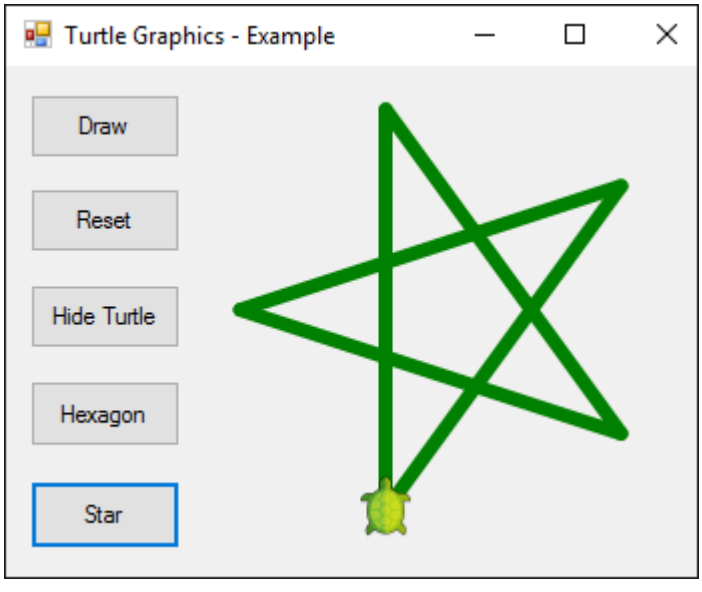
Hint:

Repeat 6 times the following in a loop:

- 60 degrees rotation.
- Forward step of 100.

Problem: * Draw a Star with the Turtle

Add a [Star] button that draws a star with 5 beams (pentagram), as on the figure below:



Hint:

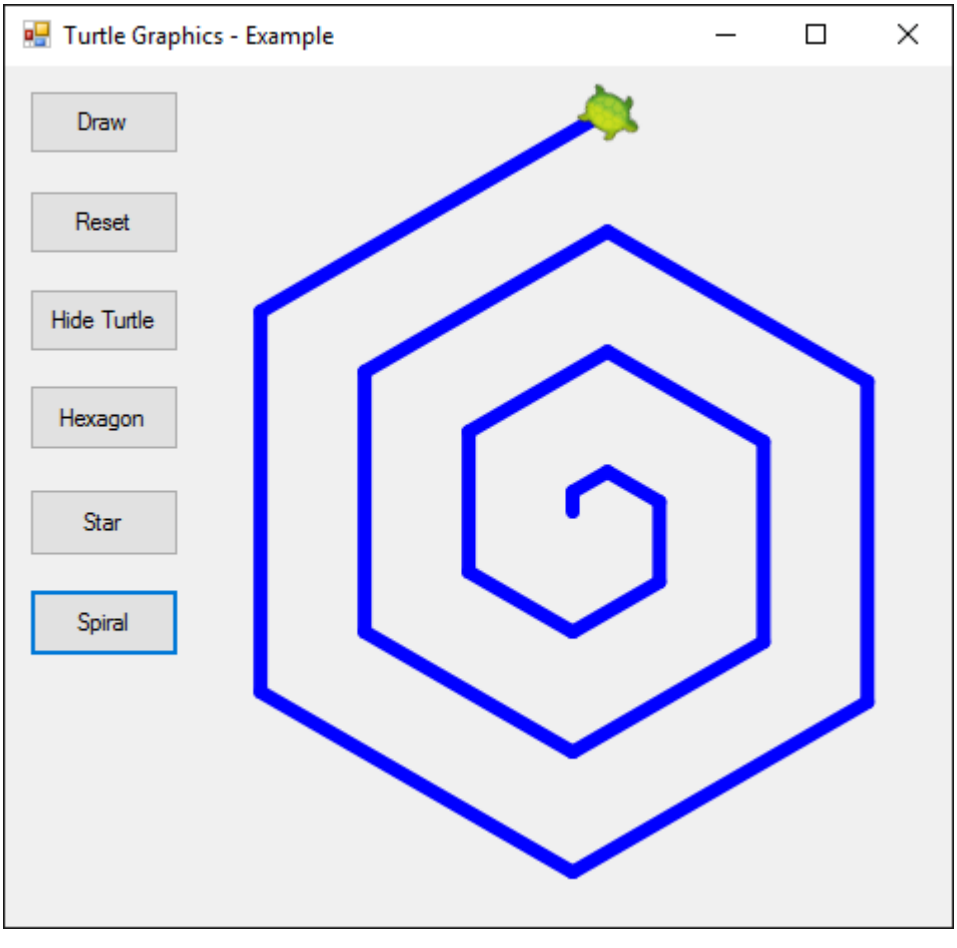
Change the color: `Turtle.PenColor = Color.Green`.

Repeat 5 times the following in a loop:

- Forward step of 200.
- 144 degrees rotation.

Problem: * Draw a Spiral with the Turtle

Add a [Spiral] button that draws a spiral with 20 beams, as on the figure below:

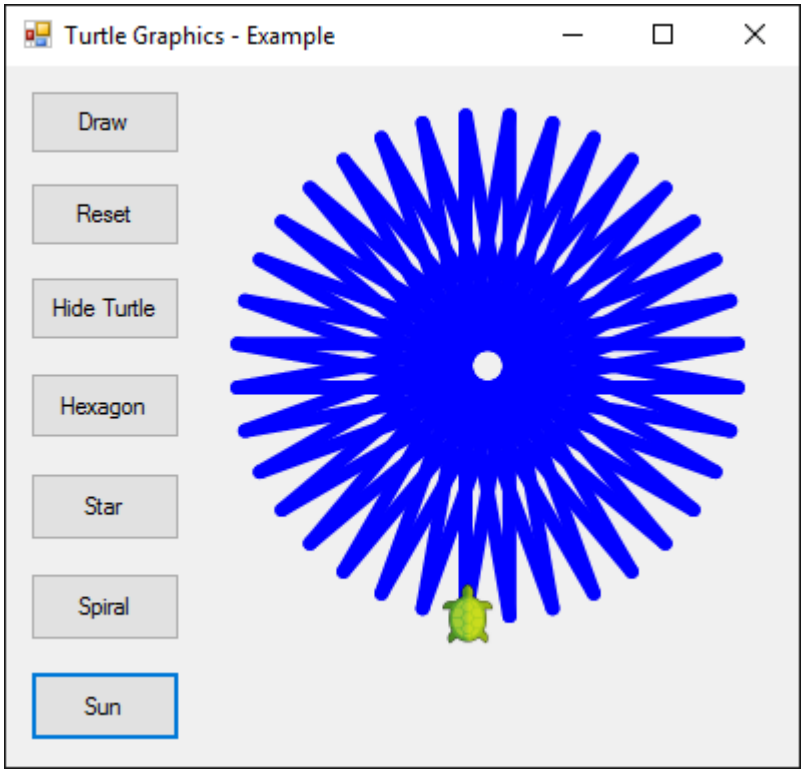


Hint:

Draw in a loop by moving ahead and rotating. In each step, decrease gradually the length of the forward step and rotate 60 degrees.

Problem: * Draw a Sun with the Turtle

Add a [Sun] button that draws a sun with 36 beams, as on the figure below:

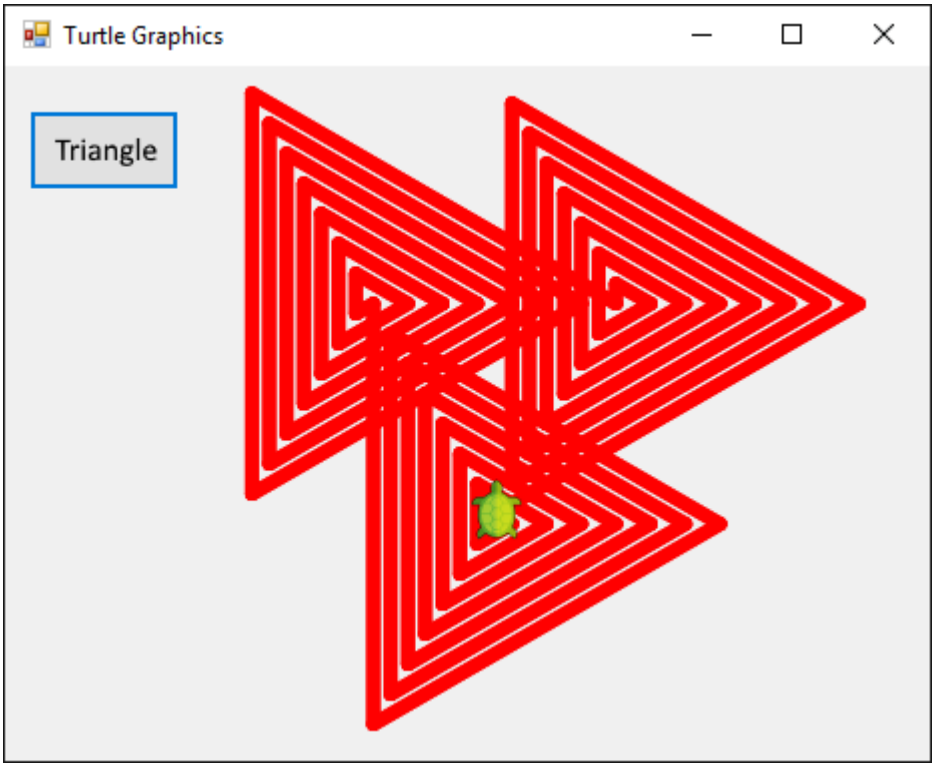


Hints:

- The entire circle consists of 360 degrees = 36 * 10 degrees
- Think about how many times you will move the turtle forward, then rotate, then move the turtle back, etc.

Problem: * Draw a Spiral Triangle with the Turtle

Add a [Triangle] button that draws three triangles with 22 beams each, as on the figure below:



Hint:

Draw in a cycle by moving forward and rotating. In each step, increase the length of the forward step with 10 and rotate 120 degrees. Repeat 3 times for the three triangles.

What's Next?

The "Programming Basics" series consists of publications for beginners in programming that covers the following topics:

1. First Steps in Programming – commands, programs, C#, Visual Studio,
2. Simple Calculations – variables, calculations, console input / output
3. Simple Conditions – conditional statements, the "if-else" construction
4. More Complex Conditions – nested if-else, logical "and", "or", "not"
5. Repetitions (Loops) – simple for-loops (repeat from 1 to n)
6. Nested Loops – nested loops and problem solving, drawing 2D figures
7. More Complex Loops – loops with a step, infinite loops with breaks
8. How to Become a Software Engineer?

The next topics are coming. Be patient!

Sign-Up to Study in SoftUni

The easiest way to become a software engineer is to go through the "Software University" training program at SoftUni. Signup now for the free Programming Basics training course: