Learning Objectives: For Loops

- **Explain `for` loop syntax**
- **Identify the causes of an infinite loop**
- **Identify the relationship between patterns, loops, and output**

For Loops

For Loop Syntax

Before you can start writing a **loop**, you need to be able to identify recurring patterns. Let's take something simple:

```
cout << "Hello" << endl;
cout << "Hello" << endl;
cout << "Hello" << endl;
cout << "Hello" << endl;
cout << "Hello" << endl;
```

The pattern is `cout << "Hello" << endl;`, and it is repeated five times. Since we know that the loop needs to run exactly *five* times, a `for` loop can be used. Here is how you write a `for` loop that repeats five times. Copy the code below into the text editor on the left and then click on the `TRY IT` button to see the output. You can also click on the ++Code Visualizer++ link below to see how a `for` loop works behind the scenes. If the visualizer does not boot up correctly, click on the `Refresh code` button to restart it. Use the `Forward >` and `Back <` buttons to navigate the program.

```
for (int i = 0; i < 5; i++) {
  cout << "Hello" << endl;
}
```

Code Visualizer

Like **conditionals**, `for` loops are code blocks. However, in addition to a **boolean** statement(s), you also declare, initialize, and increment a **variable** called the loop **iterator**. All of the code that will be repeated are placed between the curly braces `{}`. It is *recommended* that you **indent** the code within the curly braces, but it is not necessary for the loop to run.

Understanding the Loop Header

Copy the code below and `TRY IT`.

```
for (int i = 0; i < 5; i++) {
  cout << "Loop #" << i << endl;
}
```

Code Visualizer

The loop ran five times, but the variable `i` did not start at `1`. Instead, it started at `0`. C++ , like most programming languages, starts counting from 0 by default. C++ will continue counting up to, but not including, 5. The `i++` tells the system to continue counting up by `1` and the `i < 5` tells the system to stop counting *before* reaching 5.

challenge

What happens if you:

- Replace `"Loop #" << i` in the code above to `"Loop #" << i + 1`?
- Replace `i < 5` with `i < 6` in the loop header and change the print statement back to `"Loop #" << i`?
- Replace `i < 5` with `i <= 5` in the loop header?
- Replace `i++` with `i--` in the loop header?

Code Visualizer

▼ **Infinite Loops**

If you aren't careful, you can wind up with an **infinite loop**. This means that you have a loop that never ends. In the example above, if you change `i++` to `i-` then `i` will decrease by 1 after every iteration. This causes the loop iterator to never reach its specified value. The boolean expression continues to be true and the system continues to print until it times out. Always check your loop header to ensure that it does what you intend for it to do.

Turtle Graphics

Before continuing with loops, we are going to learn how to create graphical output with the **Turtle Graphics** library. Like a pencil on paper, the turtle object leaves a line as it moves around the screen.

Turtle Syntax

The first step is to create a **screen** for the turtle to move around in using the command `TurtleScreen` followed by a variable name to call that screen (i.e. `screen`). In parentheses after `screen`, you can specify the dimensions of the screen in terms of **width** and **height** respectively (i.e. `400, 300`). Then we can create our **turtle** using the command `Turtle` followed by a variable name for that turtle (i.e. `tina`). Finally in parentheses, we put in `screen` to associate the turtle with the screen that we created previously. The code below produces a turtle and a screen for the turtle to move around in.

```
TurtleScreen screen(400, 300); //width 400 pixels and height 300
        pixels
Turtle tina(screen); //creates a turtle named tina inside the
        screen
```

important

IMPORTANT

You may have noticed that there are additional lines of code within the file in the text editor to your left. It is very **IMPORTANT** that you **DO NOT** edit the header as that will cause the program to run incorrectly.

```
#include <iostream>
#include "CTurtle.hpp"
#include "CImg.h"
using namespace cturtle;
using namespace std;
```

The header above enables you to use the Turtle Graphics library as well as the C Image library. Thus, the header should never be altered.

Turtle Commands

In order to view the turtle object, it is not enough just to create it. You must give instructions to the turtle object in order for it to "move" around the screen. Here is a list of basic turtle commands that you can give to `tina` the turtle object:

Command	Parameter	Description
`tina.forward(n)`	Where `n` represents the number of pixels	Move the turtle forward
`tina.backward(n)`	Where `n` represents the number of pixels	Move the turtle backward
`tina.right(d)`	Where `d` represents the number of degrees	Turn the turtle to the right
`tina.left(d)`	Where `d` represents the number of degrees	Turn the turtle to the left

Let's try this very simple command below. Copy it into the text editor on your left and then click the `TRY IT` button to see the graphical output.

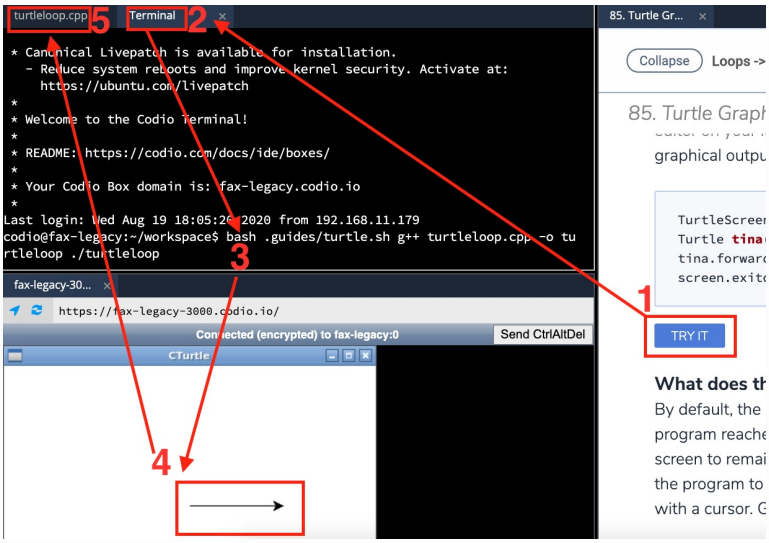
```
TurtleScreen screen(400, 300);
Turtle tina(screen);
tina.forward(100);
screen.exitonclick();
```

What does the `screen.exitonclick()` command do?

By default, the screen will close itself automatically once the program reaches the end of the code. However, if you want the screen to remain open, you can use `screen.exitonclick()` to tell the program to keep the screen open until the screen is clicked with a cursor. Go ahead and try clicking on the screen.

Turtle Output

Below is an image highlighting what happens after the `TRY IT` button is clicked.



.guides/img/TurtleGraphicsFlow

1. `TRY IT` button is clicked by the user.
2. The `Terminal` tab is opened.
3. The terminal runs the command to compile the program and to display the graphical output.
4. The output is displayed as a screen on the bottom left panel. You can click the screen to close the output.
5. Click on the `turtleloop.cpp` tab to go back to the text editor if you want to make changes to the program.

Turtle Coding: For Loop

Customize Your Turtle

You may choose to change the dimensions of your turtle screen whenever you'd like. Also, the following table provides additional commands you can use to customize `tina` the turtle.

Command	Parameter	Examples
`tina.pencolor({"COLOR"})`	Where `COLOR` represents the track or line color you want tina to leave behind	red, orange, yellow, green, blue, purple
`tina.width(W)`	Where `W` represents how wide (in pixels) tina's track is	any positive integer (e.g. 1, 10, 123, etc.)
`tina.shape("SHAPE")`	Where `SHAPE` represents the shape tina takes	triangle, indented triangle, square, arrow
`tina.speed(SPEED)`	Where `SPEED` represents how fast tina moves	TS_FASTEST, TS_FAST, TS_NORMAL, TS_SLOW, TS_SLOWEST

Turtle Challenges

Now that you know how to customize `tina`, try to recreate the images you see below using your knowledge of `for` loops.

Challenge 1

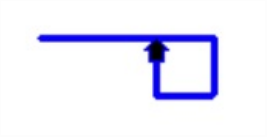
.guides/img/TurtleChallenge1

▼ **Hint**

There are multiple ways to accomplish this task but the trick lies within finding the **pattern** and then repeating it a **specific number of times**. One pattern in particular is to:

1. Go forward (creating a long line).
2. Make a right turn.
3. Go forward (creating a small line).
4. Make a right turn.
5. Go forward (creating another small line).
6. Make a right turn.
7. Go forward (creating a final small line).
8. Repeat steps #1 through #7 three more times for a total of **four** iterations.

The pattern should look something like this:



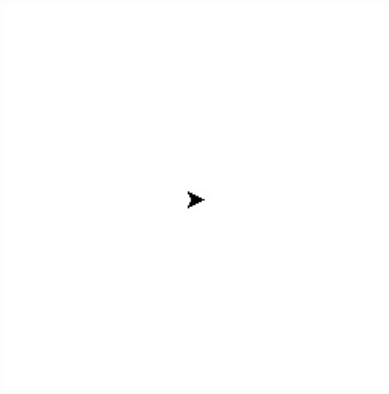
Challenge 2

.guides/img/TurtleChallenge2

▼ **Hint**

Since a circle has 360 degrees, you will need a loop that repeats 360 times. Be careful about how far the turtle moves forward and turns. The circle can get very big, very quickly.

Challenge 3



.guides/img/TurtleChallenge3

▼ **Hint**

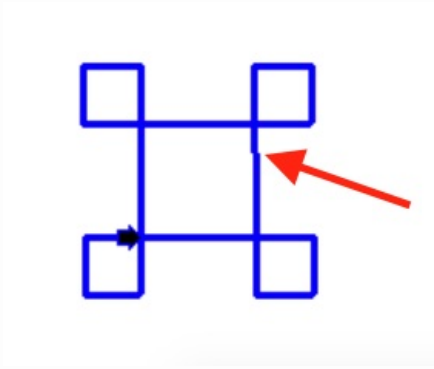
The pattern here is to move forward and make a right turn.

The trick lies within the fact that the distance the turtle moves has to get larger as the loop advances. Think of some operators that you can use to make the loop iterator variable get bigger during each iteration.

info

NOTE

Due to the dynamic and graphical nature of the Turtle Graphics library, **jagged** lines and **spotty** pixels may appear randomly as the output is being drawn. This is completely **normal**!



.guides/img/CppJaggedLine

▼ **Still having trouble with creating the outputs above?**

Here are some sample solutions:

```
tina.pencolor({"blue"});
tina.width(2);
tina.shape("arrow");
tina.speed(TS_SLOWEST);

for (int i = 0; i < 4; i++) {
  tina.forward(75);
  tina.right(90);
  tina.forward(25);
  tina.right(90);
  tina.forward(25);
  tina.right(90);
  tina.forward(25);
}
```

```
tina.pencolor({"red"});
tina.width(2);
tina.shape("square");
tina.speed(TS_FASTEST);

for (int i = 0; i < 360; i++) {
  tina.forward(1);
  tina.right(1);
}
```

```
tina.pencolor({"green"});
tina.width(2);
tina.shape("triangle");
tina.speed(TS_NORMAL);

for (int i = 10; i <= 200; i+=10) {
  tina.forward(i);
  tina.right(90);
}
```