

The **Official**
ROBLOX
Guide

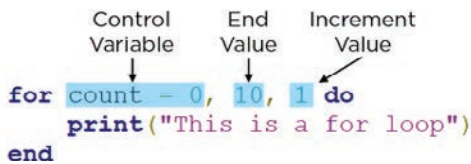
Coding with Roblox Lua

The **Official**
ROBLOX
Guide

Coding with Roblox Lua

in **24**
Hours

- **Control variable:** Tracks the current value. The assigned value marks the starting place. A control variable can be any acceptable variable name. Like other variable names, a control variable name should be clear and descriptive about what the `for` loop is doing.
- **End or goal value:** The value at which the loop should stop running. The script checks the control variable against the end value before starting the next loop.
- **Increment value:** The amount by which the control variable changes every time. Positive increment values count up; negative increment values count down.



The diagram shows a code snippet for a `for` loop with three annotations above it. The first annotation, 'Control Variable', has an arrow pointing to the variable 'count'. The second annotation, 'End Value', has an arrow pointing to the number '10'. The third annotation, 'Increment Value', has an arrow pointing to the number '1'. The code is as follows:

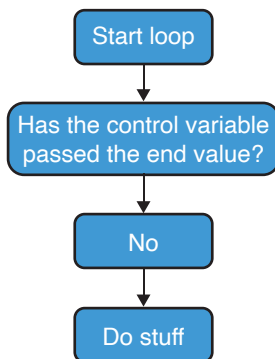
```
for count = 0, 10, 1 do
    print("This is a for loop")
end
```

FIGURE 8.3

The three values that control how many times a `for` loop runs are the control value, the end value, and the increment value.

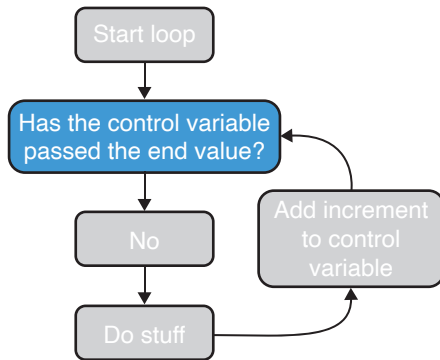
Beginning at the initial value of the control variable, the `for` loop counts toward the ending goal value, stopping once the goal value is reached:

1. The `for` loop compares the control variable with the end value. (See Figure 8.4.)

**FIGURE 8.4**

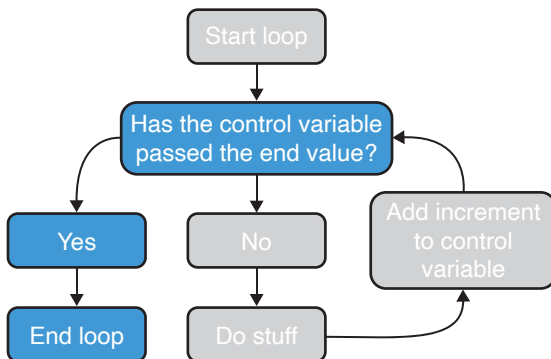
Before executing the code in the loop, the control variable is checked against the goal value.

2. After running the code, the increment value is added to the control variable. The loop then checks the control variable and starts over. (See Figure 8.5.)

**FIGURE 8.5**

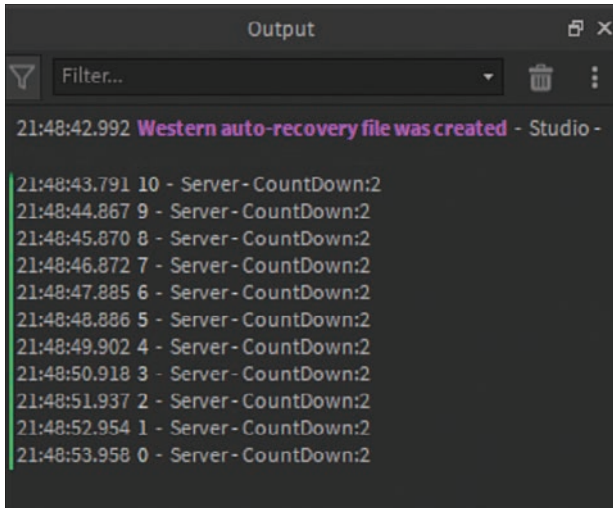
At the end of the loop, the increment value is added to the control variable.

3. Once the control variable passes the end value, the loop will stop. For example, if a loop has an end value of 10, once the control variable has passed 10, the `for` loop will stop (see Figure 8.6).

**FIGURE 8.6**

This is the flow of a complete `for` loop process.

Let's take another look at the Output shown in the Try It Yourself, displayed in Figure 8.7.

The image shows a screenshot of an IDE's 'Output' window. The window has a title bar with 'Output' and standard window controls. Below the title bar is a filter input field with the text 'Filter...'. The main area of the window displays a series of log messages. The first message is '21:48:42.992 Western auto-recovery file was created - Studio -' in a pinkish-purple color. Below it, there are eleven lines of text, each representing a tick of a countdown: '21:48:43.791 10 - Server - Countdown:2', '21:48:44.867 9 - Server - Countdown:2', '21:48:45.870 8 - Server - Countdown:2', '21:48:46.872 7 - Server - Countdown:2', '21:48:47.885 6 - Server - Countdown:2', '21:48:48.886 5 - Server - Countdown:2', '21:48:49.902 4 - Server - Countdown:2', '21:48:50.918 3 - Server - Countdown:2', '21:48:51.937 2 - Server - Countdown:2', '21:48:52.954 1 - Server - Countdown:2', and '21:48:53.958 0 - Server - Countdown:2'. The text is white on a dark background.**FIGURE 8.7**

This output of a `for` loop counts down every second.

The loop that ran each time a number was printed is called an iteration. An *iteration* is the complete process of checking the control value, running code, and updating the increment value. Since the count started at 0 and ended after 10, the code actually went through eleven iterations.

Keep this in mind as you design your loops. If it's important for a count to go a specific number of times, you'll probably want the starting value to be 1 instead of 0.

Increments Are Optional

If an increment value isn't included, the default value of 1 is used. The code snippet begins at 0 and counts upward to 10:

```
for countUp = 0, 10 do
  print(countUp)
  wait(1.0)
end
```

Different `for` Loop Examples

Changing the values of the control variable, end goal, and increment changes how the loop functions. The `for` loop you just wrote could instead count up to 10 or count down in odd numbers. The following are examples of `for` loops with different start, end, and increment values.

Counting Up by One

```
for count = 0, 5, 1 do
    print(count)
    wait(1.0)
end
```

Counting Up in Even Numbers

```
for count = 0, 10, 2 do
    print(count)
    wait(1.0)
end
```

Be careful not to reverse the starting and goal values, like so:

```
for count = 10, 0, 1 do
    print(count)
    wait(1.0)
end
```

If the control variable starts out beyond the end value, like in the earlier example, the `for` loop doesn't run at all. In this case, the `for` loop is counting up and checking if `count` is greater than 0. When the `for` loop does its first check, it sees that 10 is greater than 0, so it stops the loop without printing anything.

▼ TRY IT YOURSELF

In-World Countdown

So far, messages have only been displayed within the Output window. Now it's time to start communicating information to people in your environments. In this Try It Yourself, you use a graphical user interface (GUI) to display information where everyone can see it. GUIs are like sticker labels that can be used to display information within the world.

Setup

For the setup, you create a `SurfaceGui` and `TextLabel` and size them to the part to display the countdown. Since this is a coding book, we won't get too much into how these work. If you want to know more, you can find more detailed explanations on the Roblox Developer Hub:

1. Create a new part.
2. Insert a `SurfaceGui` object into the part. Nothing obvious happens, but `SurfaceGui` objects act as containers for anything you want to display.

3. Select SurfaceGui and insert a TextLabel object. This displays the actual text. (See Figure 8.8.)

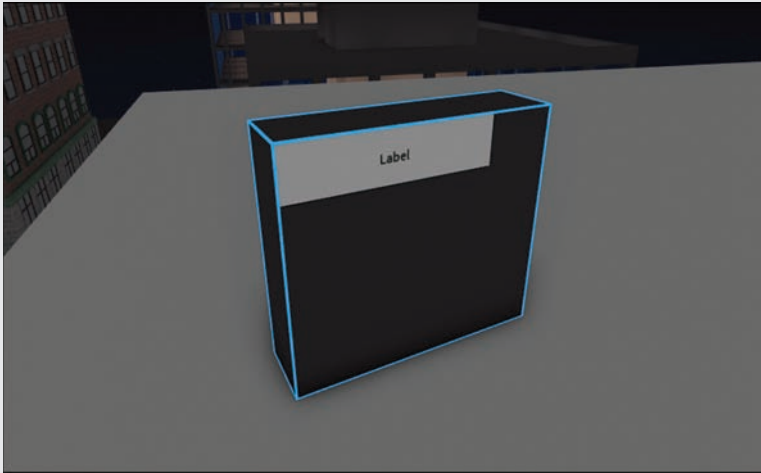


FIGURE 8.8

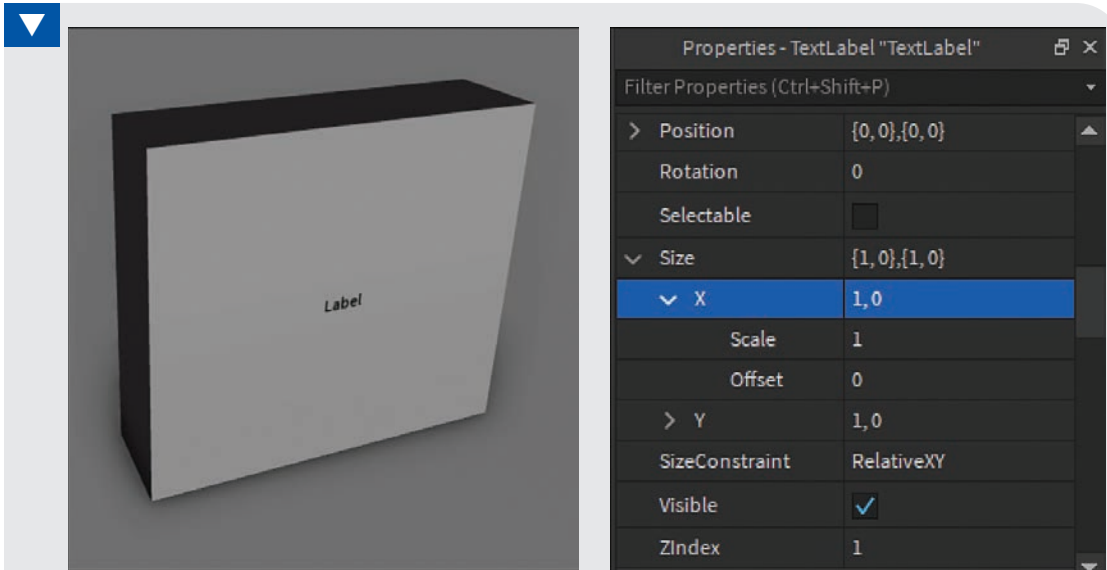
The TextLabel is added on the front of a part.

TIP

Finding the TextLabel

If you can't see the TextLabel, it probably appeared on a different side of the part. You can rotate the part or change the SurfaceGui's Face property to fix it.

4. Select the TextLabel. In Properties, expand Size. For X Scale, type 1, and in Offset, type 0. Do the same for Y. This should make the TextLabel take up the entire side of the part. (See Figure 8.9.)

**FIGURE 8.9**

The TextLabel takes up the entirety of the side.

5. Still in TextLabel's properties, scroll almost to the bottom to TextScaled and enable it. This sizes the font to fit as shown in Figure 8.10.

**FIGURE 8.10**

The text is automatically scaled to fit the entire TextLabel.