



The Big Idea:

This lesson simplifies the control of digital pins by assigning the pin numbers to an integer variable and by calling the `digitalWrite` command multiple times by means of a `for` loop.

Background:

Lesson 4 introduced digital pins. The sketches written turned light-emitting diodes on and off. The program code in those sketches has two characteristics that, while appropriate for an early lesson, are considered inefficient. These are:

1. The same programming statement appears multiple times, as shown in Example 7-1.

Example 7-1. `digitalWrite`

```
digitalWrite( 2, HIGH);  
digitalWrite( 3, HIGH);  
digitalWrite( 4, HIGH);  
digitalWrite( 5, HIGH);  
digitalWrite( 6, HIGH);  
digitalWrite( 7, HIGH);
```

2. The pin number parameter (2, 3, 4, 5, 6, and 7 in Example 7-1) is an integer **literal**. Good practice is to replace it with an integer **variable**.

A `for` loop is a way of performing a set of tasks a fixed number of times. Consider the host of a child's birthday party. At the end of the party each child is thanked for his or her gift, given a gift bag to take home, thanked for coming, sent out the door to be met by a parent. This is a kind of `for` loop that can be written as a set of instructions as follows:

For each child:

- Thank for the gift.
- Hand a gift bag.
- Thank for coming.
- Escort through door to parent.

When writing sketches for the Arduino,[™] the programmer often needs to perform a set of programming statements a fixed number of times. In the example of the LEDs shown here, there is only one statement, the one that turns the LED on by setting the output `HIGH`. The process is:

For each digital pin:

- Set the output HIGH

For that matter, in the `setup()` method a `for` loop can be used to set the digital pins to output mode:

For each digital pin:

- Set each pin to OUTPUT

Table 7-1. Vocabulary

Term	Definition
comparison operator	Also called logical operators, these are used in logical tests. The C language has six: > greater than < less than >= greater than or equal to <= less than or equal to == equal to != not equal to
curly brace { }	A symbol that indicates the beginning and ending of a group of C-language statements. The beginning of the group is indicated by the { symbol, the ending by the }. These are referred to as the <i>opening</i> curly brace and the <i>closing</i> curly brace.
for loop	A type of loop in which the number of times the statements in a loop run is numerically defined.
increment	The addition of the value 1 to some integer variable. For example, if <code>myNumber</code> is the name of an integer variable and the current value is 6, the increment of <code>myNumber</code> results in replacing that value with 7.
logical test	A logical expression that evaluates to either true or false. Both terms are C-language keywords. For example, if <code>myNumber</code> is an integer variable to which a value has been assigned, a logical statement that must evaluate to true or false is: <code>myNumber > 6</code> This statement is either true or false.
loop	A group of C-language programming statements enclosed in a set of curly braces. This group is intended to be run multiple times.

Description:

The for loop concept

The **for loop** is used to execute a set of programming statements multiple times. It consists of two parts. These are the set of programming statements to be run and the **for** loop programming statement that sets the criteria for allowing them to be run and determining when execution should stop.

As illustrated in Figure 7-1, the programming statements to be executed are contained within a set of **curly braces** `{}` immediately following the **for** loop statement.

The **for** loop statement itself has three parts. Each places a role in setting up and running a logical test that determines how many times the statements are executed. These are the initial condition, the test, and the increment. Each contains one or more Java programming statements.

Eventually the programming statements in increment will create a situation where the test no longer evaluates to true. When this happens, the programming statements within the curly braces are no longer executed, and the program will move to what comes after the **for** loop.

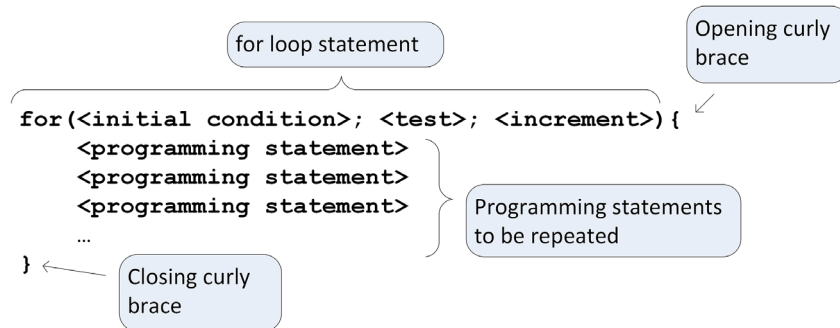


Figure 7-1. Parts of the for loop statement

The **for** loop statement itself has three parts. The underlying idea is that an integer counter is created to keep track of the number of times the programming statements are run. Each time they run is called a **loop**. After each loop, the counter changes somehow. This is the **increment**. After the increment, the counter is tested to see if the loop should run again.

Suppose, for example, someone you know has enrolled in a public speaking course. As part of this course, she is to perform the following set of tasks five times:

- Locate a complete stranger.
- Introduce herself.
- Learn something about this stranger.
- Record the stranger's name and what was learned.

This set of tasks to be completed for each stranger is the real world equivalent of the programming statements to be repeated. Suppose, further, that this person wants perform her tasks in an organized manner. She might then take the following approach:

Step 1: Before beginning, write the number 1 on a piece of paper. This is a counter and reflects which attempt she is about to undertake. This is the initial condition. Her first attempt is attempt number 1. So the counter is set to 1.

Step 2: Examine the counter on the piece of paper. Is it less than or equal to 5? If yes, she isn't done and must make one more attempt. But if it is greater than 5, then she has completed five attempts and will stop.

Step 3: Perform the four tasks: locate, introduce, learn, and record.

Step 4: Increment the counter by adding 1 to it and recording the new value.

Step 5: Go to Step 2.

The for loop in C

Referring back to Lesson 3, suppose the `setup()` method is to initialize pins 2 through 7 to the `OUTPUT` mode so that the attached LEDs can be turned on and off in the `loop()` method. Instead of having six separate and nearly identical `pinMode` statements, a `for` loop can be used to simplify the initialization as shown in Example 7-2:

Example 7-2. Simplifying code using a for loop

```
for(int pinNum = 2; pinNum <= 7; pinNum = pinNum + 1){  
    pinMode(pinNum, OUTPUT);  
}
```

These statements instruct the Arduino™ to do the following:

Step 1: Create an integer variable to represent the pin being addressed. This is the counter. Initialize it to the number of the first pin to be set to `OUTPUT`.

Step 2: Have all the pins through pin 7 been configured? If yes, quit.

Step 3: Initialize the pin represented by the current value of `pinNum` to the `OUTPUT` mode.

Step 4: Increment `pinNum` by adding 1 to it.

Step 5: Go to Step 2.

Comparison operators

The portion of the `for` loop that tests to see if the loop should be run again is shown in Example 7-3.

Example 7-3.



```
pinNum <= 7;
```




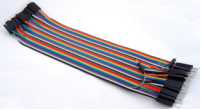
This is a comparison of the value of the variable `pinNum` with the integer literal 7. The result of this comparison is either true or false. The symbols `<=` combination is called a *comparison operator*. This particular one is read as, "less than or equal to" and is one of six comparison operators in the C language. The set of six comparison operators appears in Table 7-2.

Table 7-2. Comparison operators

Operator	Symbol	Example	Meaning
Greater than	>	<code>a>b</code>	True if the value of a is greater than the value of b. Otherwise false.
Greater than or equal to	>=	<code>a>=b</code>	True if the value of a is greater than the value of b or if a is equal to b. Otherwise false.
Less than	<	<code>a<b</code>	True if the value of a is less than the value of b. Otherwise false.
Less than or equal to	<=	<code>a<=b</code>	True if the value of a is less than the value of b or if a is equal to b. Otherwise false.
Equal to	==	<code>a==b</code>	True if the value of a is equal to the value of b. Otherwise false.
Not equal to	!=	<code>a!=b</code>	True if the value of a is not the same as the value of b. Otherwise false.

Materials:

Quantity	Part	Image	Notes	Catalog Number
1	Arduino™ Uno		Single-board computer. This board is delicate and should be handled with care. When you are not using it, keep it in a box or plastic bag.	3102
1	USB Cable		This is a standard USB adapter cable with a flat connector on one end and a square connector on the other.	2301
1	Computer with at least one USB port and access to the Arduino™ website, http://www.arduino.cc	---	The operating system of this computer must be Windows, Macintosh OS/X, or Linux.	---

Quantity	Part	Image	Notes	Catalog Number
6	Light-emitting diodes (LEDs)		Single color, about 0.02 amps rated current, diffused.	1301
6	220 ohm resistors		¼ watt, 5% tolerance. Color code: red-red-brown-gold.	0102
1	Bread-board		Used for prototyping.	3104
As req'd	Jumper wires		Used with bread-boards for wiring the components.	3105

Procedure:

Part 1: Explore for loops

1. Open the Arduino™ IDE. Create a new Arduino™ sketch and name it Lesson7ForLoopsText.
2. Add the header and the `setup()` method.

Snippet 7-1.

```

/* Lesson7ForLoopText
   <author>
   <date>
*/

void setup(){
  Serial.begin(9600);
}

```

3. Insert the simple `for` loop into the `loop()` method as shown in Snippet 7-2.

Snippet 7-2.

```
...  
void loop(){  
    Serial.println(); // insert a blank line  
  
    Serial.println("Beginning loop");  
    for(int counter = 1; counter <= 6; counter++){  
        Serial.println("Hello, world!");  
    }  
  
    Serial.println("Finished loop");  
    while(true); // stop here  
}
```

7

Open the Serial Monitor. It should show the String `Hello, world!` six times. Above and below should be the messages indicating the loop is about to start and that it has finished.

```
Beginning loop  
Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!  
Finished loop
```

4. For step 3, the programming statement inside the `for` loop is printed the first time when `counter = 1`, the second time when `counter = 2`, the third time when `counter = 3`, and so on until the counter is equal to 6. When the counter is next incremented, it equals 7, and the test for the `counter <= 6` fails. The loop stops, and the program execution jumps to the next statement after the `for` loop's curly braces.

Remember that `counter` is an `integer` variable. Its value can be printed from inside the loop. Verify this by replacing the programming statement inside the `for` loop curly brace as shown in Snippet 7-3.

Snippet 7-3.

```
...  
void loop(){  
    Serial.println(); // insert a blank line  
  
    Serial.println("Beginning loop");  
    for(int counter = 1; counter <= 6; counter++){  
        Serial.print("The counter is: ");  
        Serial.println(counter);  
    }  
  
    Serial.println("Finished loop");  
    while(true); // stop here  
}
```

The Serial Monitor will now show:

```
Beginning loop  
The counter is: 1  
The counter is: 2  
The counter is: 3  
The counter is: 4  
The counter is: 5  
The counter is: 6  
Finished loop
```

5. The counter does not have to begin at 1. Nor does it have to count up. Nor does the increment value have to be 1. Modifying the `for` loop to read as shown in Snippet 7-4 has the counter counting down from 20 by 3's.

Snippet 7-4.

```
...  
void loop(){  
    Serial.println(); // insert a blank line  
  
    Serial.println("Beginning loop");  
    for(int counter = 20; counter >= 0; counter -= 3){  
        Serial.print("The counter is: ");  
        Serial.println(counter);  
    }  
}
```



```
Serial.println("Finished loop");
while(true); // stop here
}
```

The Serial Monitor will display:

```
Beginning loop
The counter is: 20
The counter is: 17
The counter is: 14
The counter is: 11
The counter is: 8
The counter is: 5
The counter is: 2
Finished loop
```

6. Save and close the Arduino™ sketch.

Part 2: Controlling digital pins with for loops

The following steps require the same bread-board configuration used for Lesson 4.

1. Connect the LEDs to their current limiting resistors and these, in turn, to the Arduino.™

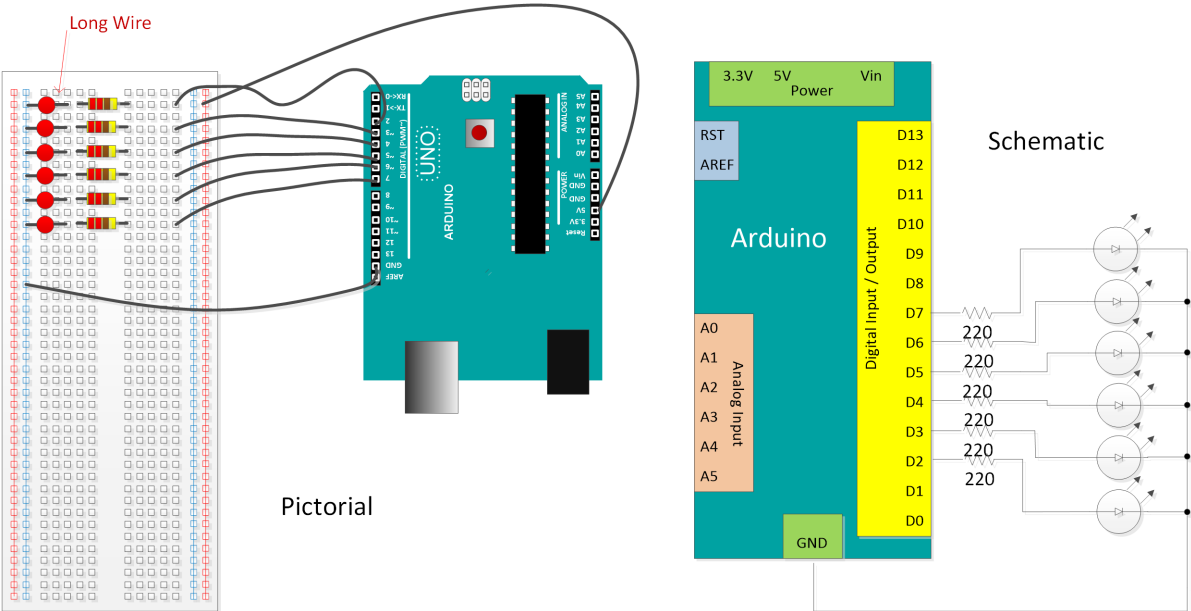


Figure 7-2. Pictorial and schematic diagrams of connection of LEDs to Arduino™

2. Create a new Arduino™ sketch. Name this one **Lesson7ForLoopsLED**. The header should be similar to that for the previous sketch but with the new sketch name and date.
3. Below the header comments, add the `setup()` method. It will take advantage of a `for` loop to simplify initializing the modes of the digital pins. Notice the counter begins at 2 because this is the first digital pin wired to an LED. It ends by initializing pin 7 because that is the last digital pin connected to an LED.

Snippet 7-5.

```
/* Lesson7ForLoopsLEDv
   <author>
   <date>
*/

void setup(){
    for(int pinNum = 2; pinNum <= 7; pinNum++){
        pinMode(pinNum, OUTPUT);
    }
}
```

4. Now add a `loop()` method that will light each of the LEDs in pin number order and then extinguish them in reverse order. It should look something like Snippet 7-6.

Snippet 7-6.

```
...
void loop(){
    // turn on in ascending order
    for(int pinNum = 2; pinNum <= 7; pinNum++){
        digitalWrite(pinNum, HIGH);
        delay(500); // wait 1/2 second
    }

    // turn off in descending order
    for(int pinNum = 7; pinNum >= 2; pinNum--){
        digitalWrite(pinNum, LOW);
        delay(500);
    }
}
```

Upload the sketch to the Arduino.™ The LEDs should now be lighting and going out much like a thermometer going up and down. Take time to understand how the `for` loops are being used to accomplish this.

5. Comment out the `delay(500)` programming statement inside the turn off `for` loop. Then upload the sketch. The lights should still come on in order but appear to all go off at once. In fact, they are going off in turn, just as before, but so quickly the actions appear to be simultaneous.
6. Save the sketch.

Exercises:

Exercise 7-1. Light LEDs one at a time

Write an Arduino™ sketch (name it `Lesson7Exercise1`) that uses a `for` loop to light each LED in turn, beginning with the one wired to pin 2 and ending with the LED wired to pin 7. But only one LED is lit at a time. The result should look like a light moving from LED to LED, from pin 2 through pin 7. Set the delay between each LED to 1/10 second.

Exercise 7-2. Odds and evens

Write an Arduino™ sketch (name it `Lesson7Exercise2`) that uses `for` loops to alternately flash the LEDs attached to even-numbered pins (2, 4, 6) and LEDs attached to odd-numbered pins (3, 5, 7). The interval should be 1/10 second.

Exercise 7-3. LEDs move in pairs

Write an Arduino™ sketch (name it `Lesson7Exercise3`) that has a pair of LEDs light up so that they appear to move as a pair from pin 2 through pin 7. The interval should be 1/10 second. The lighting pattern is:

```
pin 2 HIGH
pins 2 and 3 HIGH
pins 3 and 4 HIGH
pins 4 and 5 HIGH
pins 5 and 6 HIGH
pins 6 and 7 HIGH
pin 7 HIGH
all off
pattern repeats
```

