

What Will You Learn?

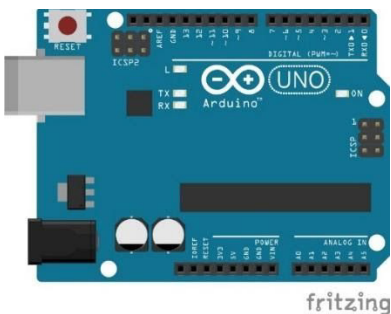
Learn how to connect a Servo motor to the microcontroller and write code to control the motor's movement.

Why Should You Learn This?

If you have gone through the LED Circuit and LED Blink exercises, you are ready to move on to programming a Servo. A Servo is a motor that can be programmed to move. There are 2 different types of Servos that are typically used, a full rotation (or continuous) Servo and a standard (or position based) Servo. The same code works for both types of Servos. If you don't know which kind you have, you'll find out once you connect it to the microcontroller. If it spins all the way around, it is a full rotation Servo. If it only moves to a certain position, it is a standard Servo.

Here's What You'll Need:

Arduino Microcontroller



Servo



Red Wire



Black Wire



White Wire

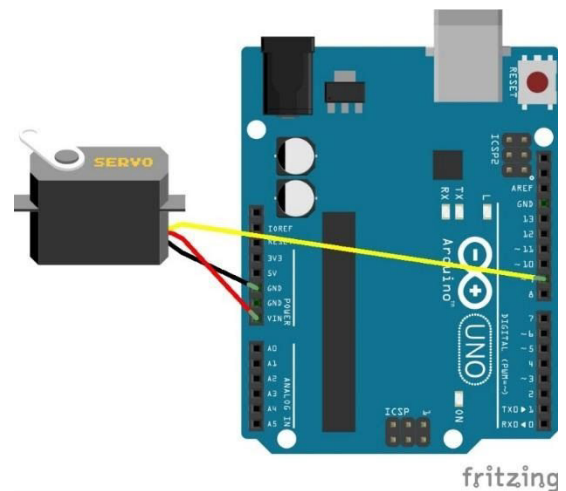


Safety First!

Never allow the red and black wires to touch together while they are connected to a powered microcontroller board, as that creates a short circuit. A short circuit can potentially cause the wires and/or the microcontroller to get hot enough to burn the skin. In addition, in the event of a short circuit the microcontroller can potentially catch on fire.

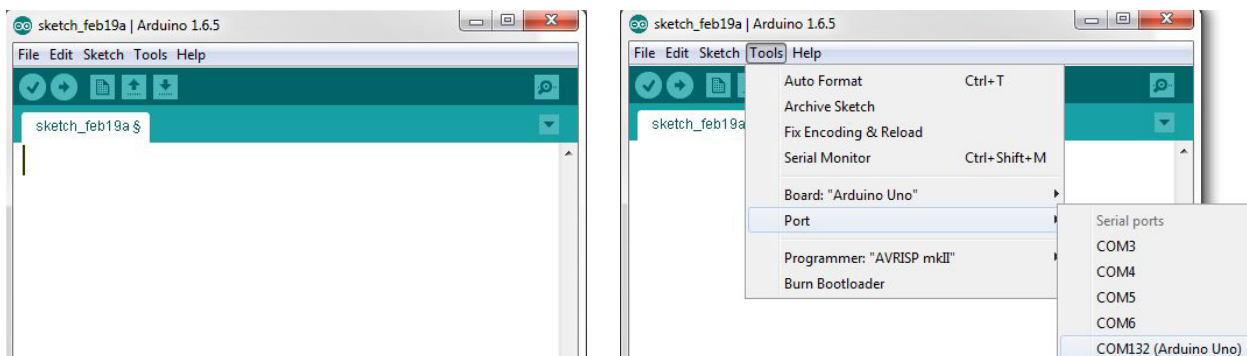
Let's Get Connected!

Servos are controlled using three wires: a positive wire (usually red), a negative wire (usually black), and a signal wire (usually yellow or white). First, connect the Servo's positive wire (red) to the Vin pin on the microcontroller. If you need to use an extra wire to do this, it is smart to use a red wire, so that it matches the colors of the wires coming off the Servo. Next, connect the Servo's ground wire (black) to the GND pin on the microcontroller. Finally, connect the Servo's signal wire (yellow or white) to pin 9 on the microcontroller (we will be using pin 9 in our code).



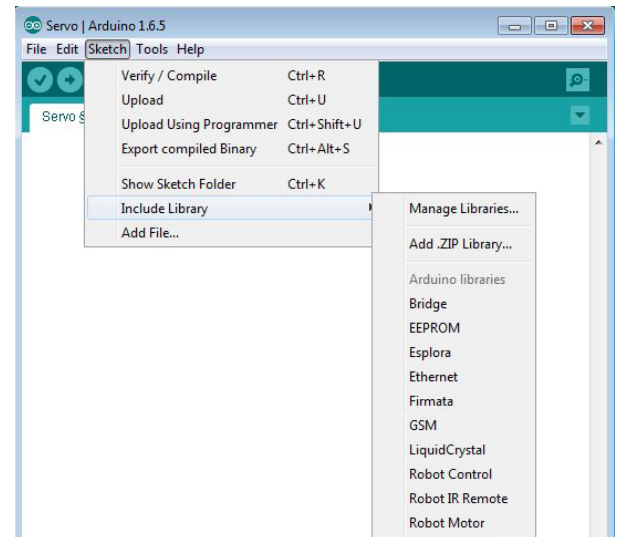
Time to Write Some Code!

1. Start by opening a new sketch and erasing everything that is already there. Don't forget to connect your microcontroller to the correct port, so that you can upload your program. (See the "Troubleshooting" document if you are having problems connecting.)

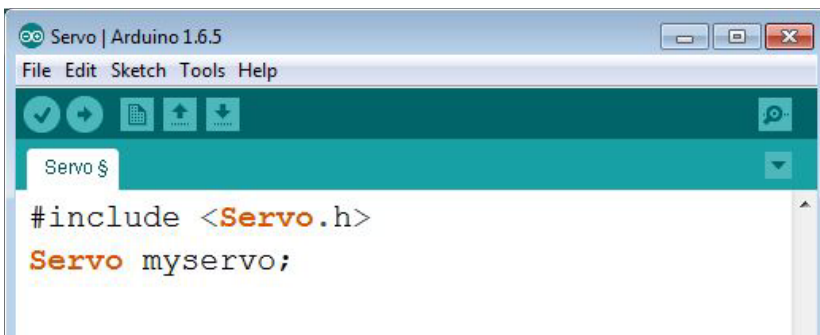


2. Writing a program that will generate the signals that you need to control a Servo is a bit complicated. Thankfully, you don't have to write this program yourself. You can use a package of pre-programmed code, called a "library", which comes with the Arduino software. Using the Servo.h library will make it much easier for you to control your Servo.

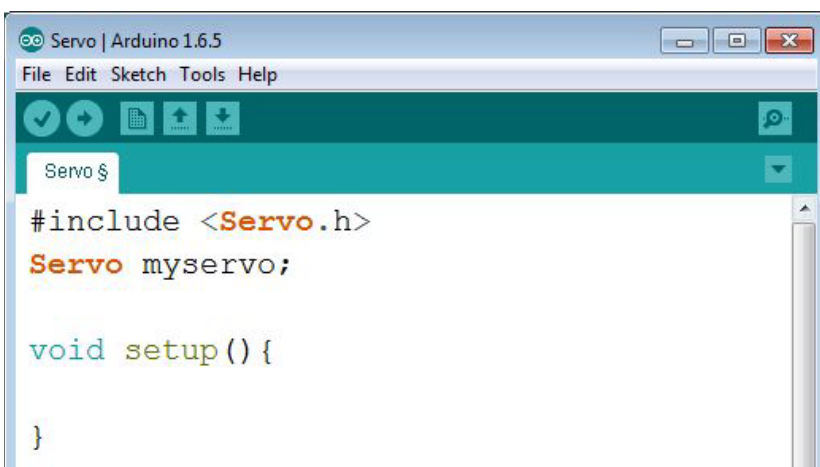
To use the library, go to Sketch → Include Library then click on "Servo". After you click on the library, it will add a line of code that should look like this: `#include <Servo.h>`. This line of code tells the computer to include the Servo library in our project.



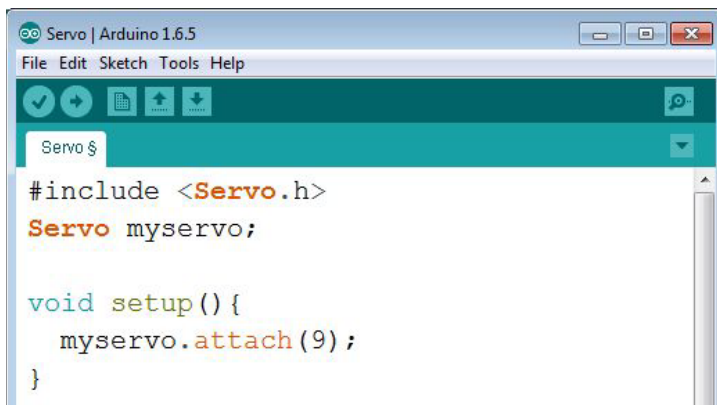
3. Use your arrow key to move to the next line in your program and then type `"Servo myservo;"`. This names the Servo in the code called "myservo".



4. Next, press Enter twice to move your cursor down, then type in `"void setup() { "` and press enter once more. When you do this, a closed bracket `"}"` should appear at the bottom of the code. These brackets are like hamburger buns, you can decide what you want to put on your burger, but you always need a top and bottom bun. Anything you type between the open bracket and the closed bracket will be part of the setup.



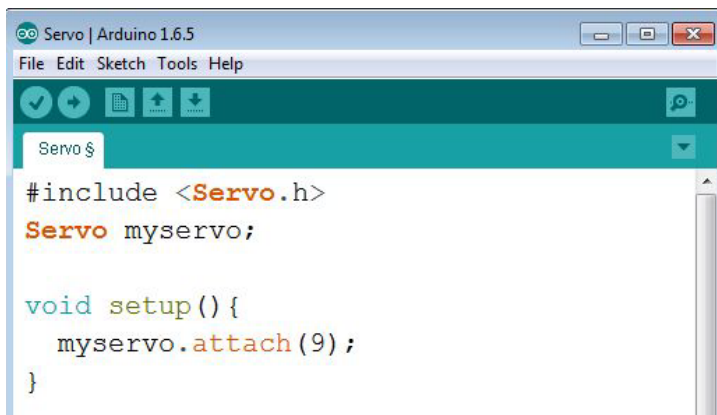
5. You need to use one pin on the Arduino microcontroller to send signals to the Servo. You can tell the microcontroller to use pin 9 (the pin you used in "Let's Get Connected") by typing `myservo.attach(9);`. Note that you use pin 9 because it has the ~ symbol next to it, which means that it generates the PWM type of signal (the type of signal needed for the Servo).

A screenshot of the Arduino IDE window titled "Servo | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, running, and other functions. The main text area shows the following code:

```
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}
```

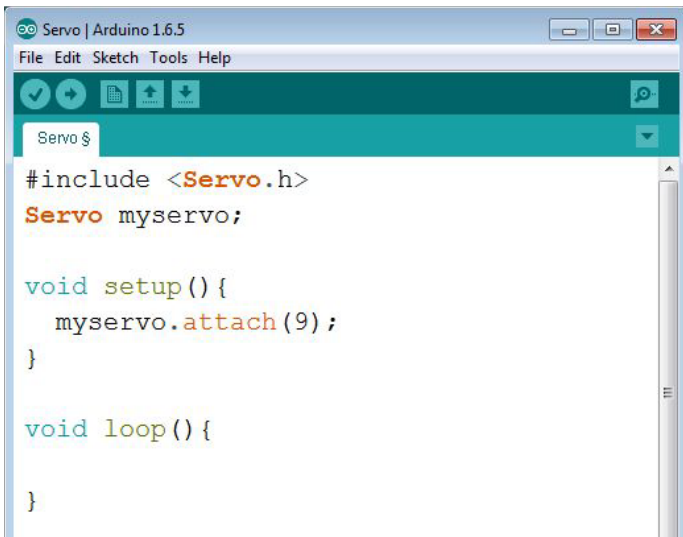
6. Use your arrow key to move to the next line in your program, and then press Enter twice. You want the closed bracket to remain where it is, but you want to move your cursor down the page.

A screenshot of the Arduino IDE window titled "Servo | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for saving, running, and other functions. The main text area shows the same code as the previous screenshot, but the cursor is now at the end of the line `myservo.attach(9);`, and a new line has been added below it:

```
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}
```

7. Next, you will code the action phase of your servo. First, type in the line `void loop() {`. You will see that, once again, when you have typed this line of code in and pressed enter, the program will automatically make a closed bracket at the bottom of the page.

A screenshot of the Arduino IDE window titled "Servo | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, and running. The main text area shows the following code:

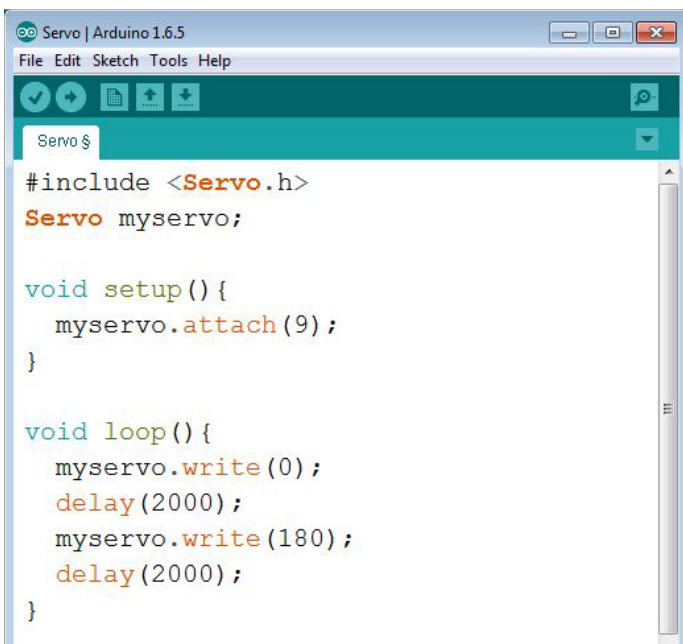
```
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}

void loop() {
}
```

8. The last part of the code will control the Servo's motion. There are 4 lines of code that will control your Servo.

- The first line, "**myservo.write**(0) ;" causes the continuous Servo to spin fast in one direction (typically to the left, but some Servos are different) and causes the standard Servo to go to 0 degrees.
- The second line, "**delay**(2000) ;" causes the microcontroller to wait before reading the next line of code, so the continuous Servo would keep spinning in one direction for 2000 milliseconds (2 seconds) and the standard Servo would stay at 0 degrees for 2000 milliseconds (2 seconds).
- The third line, "**myservo.write**(180) ;" causes the continuous Servo to spin fast in the opposite direction and causes the standard Servo to go to 180 degrees.
- The last line, "**delay**(2000) ;" causes the microcontroller to wait for 2000 milliseconds (2 seconds) before reading the next line of code, so the continuous Servo would keep spinning for 2000 milliseconds (2 seconds) and the standard Servo would stay at 180 degrees for 2000 milliseconds (2 seconds). The value that you put in "**myservo.write**() ;" controls the Servo's motion, and the "**delay**() ;" tells the microcontroller how long to wait before reading another line of code. This gives the Servo time to move.

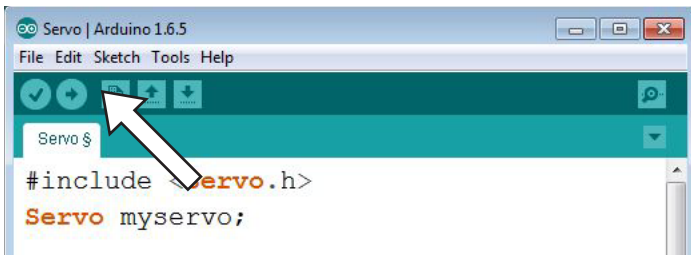
A screenshot of the Arduino IDE window titled "Servo | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, and running. The main text area shows the following code:

```
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}

void loop() {
  myservo.write(0);
  delay(2000);
  myservo.write(180);
  delay(2000);
}
```

9. Once all your code is written, upload the code to your Arduino microcontroller. If you get any errors, see the “Tips for Success!” section at the bottom of this document or the “Troubleshooting” guide.



Your Servo should now be moving. Congratulations! If you have multiple Servos, test each one and see how the same code affects the different Servos.

Tips for Success!

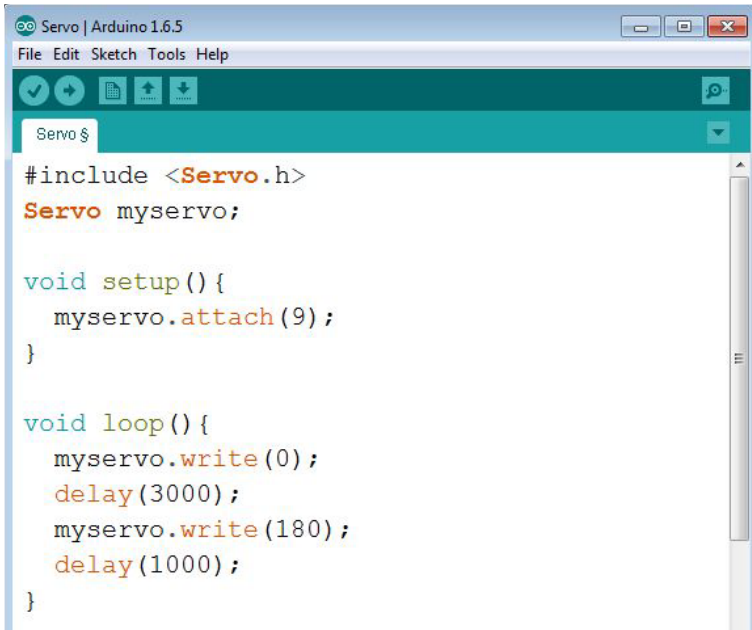
- If you get an error when you try to upload your program to the microcontroller, you can download the “Troubleshooting” document and try to find a solution. The document explains the most common problems, such as: missed semicolons, missed brackets, misspelled words, and not connecting to the board.
- If your code uploads, but the Servo does not move, check that your wires are all in the right place. Next, check that the “setup” section of your program looks exactly like the code written in step 5 of “Time to Write Some Code”. Sometimes people change the number in “`myservo.attach() ;`”, but this sends the control signal to the wrong pin on the microcontroller.

Are You Ready for Some Challenges?

1. Try to make your continuous Servo spin to the left for 3 seconds and to the right for 1 second.
2. Try to make your continuous Servo spin slow to the left forever.
3. Try to make your standard Servo go to 0° for 1 second, 45° for 1 second, 90° for 1 second, 135° for 1 second, 180° for 1 second, and then back, stopping at each of the above mentioned degrees for 1 second each. For example, 0°, 45°, 90°, 135°, 180°, 135°, 90°, 45°, 0°, etc.

Challenge Solutions:

Challenge 1 Solution:



```
Servo | Arduino 1.6.5
File Edit Sketch Tools Help

Servo $
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}

void loop() {
  myservo.write(0);
  delay(3000);
  myservo.write(180);
  delay(1000);
}
```

Challenge 2 Solution:



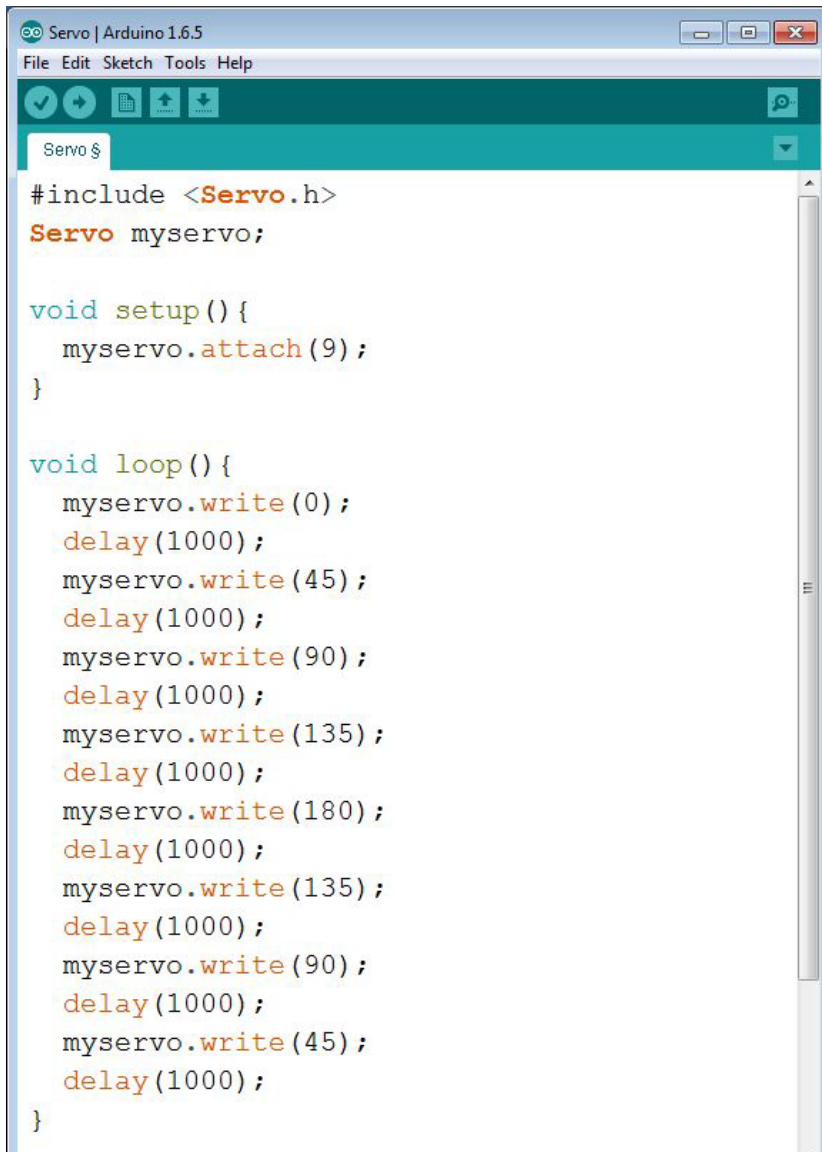
```
Servo | Arduino 1.6.5
File Edit Sketch Tools Help

Servo $
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}

void loop() {
  myservo.write(80);
}
```


Challenge 3 Solution:

A screenshot of the Arduino IDE interface. The title bar reads "Servo | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, and uploading. A status bar at the top shows "Servo \$". The main text area contains the following C++ code:

```
#include <Servo.h>
Servo myservo;

void setup() {
  myservo.attach(9);
}

void loop() {
  myservo.write(0);
  delay(1000);
  myservo.write(45);
  delay(1000);
  myservo.write(90);
  delay(1000);
  myservo.write(135);
  delay(1000);
  myservo.write(180);
  delay(1000);
  myservo.write(135);
  delay(1000);
  myservo.write(90);
  delay(1000);
  myservo.write(45);
  delay(1000);
}
```