5. Hello World! (for UNO and Arduino Every)

A "*Hello World!*" in the Arduino sphere is a blinking LED.
You just need an Arduino board and a USB cable.
Open a new file in the IDE. The lines of code below are already written. They form the basis of every program. More about that later.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Name and save it.
Then type the following text into the Arduino sketch editor but you can skip the lines starting with a `//` as they are comments.

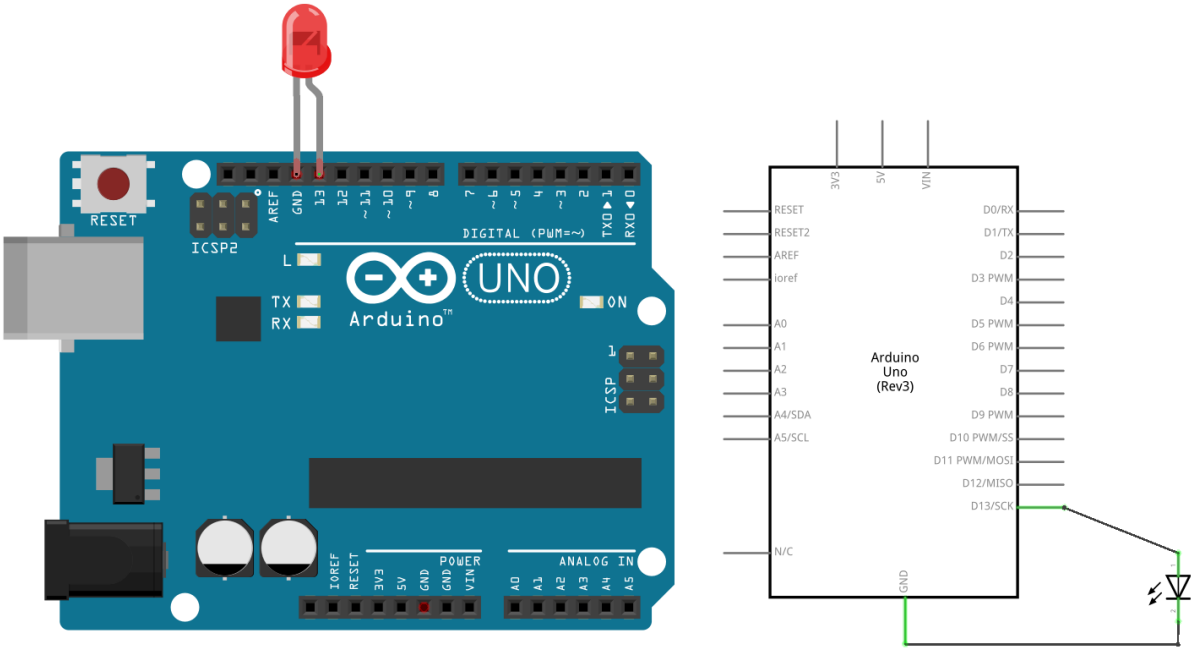
```
// LED connected to digital pin 13 > works for UNO & Every
const int ledPin = 13;

// the setup function runs once when you press reset
// or power the board
void setup() {
    // initialize digital pin 13 as an output.
  pinMode(ledPin, OUTPUT);

}

  // the loop function runs over and over
void loop() {
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(ledPin, HIGH);
  // wait for 1000 milliseconds or 1 second
  delay(1000);
  // turn the LED off by making the voltage LOW
  digitalWrite(ledPin, LOW);
  // wait for another second
  delay(1000);
}
```

Press the **Verify** button to check if your code is correct.
If everything is fine, you'll see the message **"Done compiling"** appear at the bottom of the Arduino IDE. The Arduino IDE has translated your sketch into an executable program that can be run by the board.
Now, press the **Upload button**. This will reset the board and force it to stop its current functions. Then the current compiled sketch is send to the board and got stored in its memory. Subsequently the board will run it.

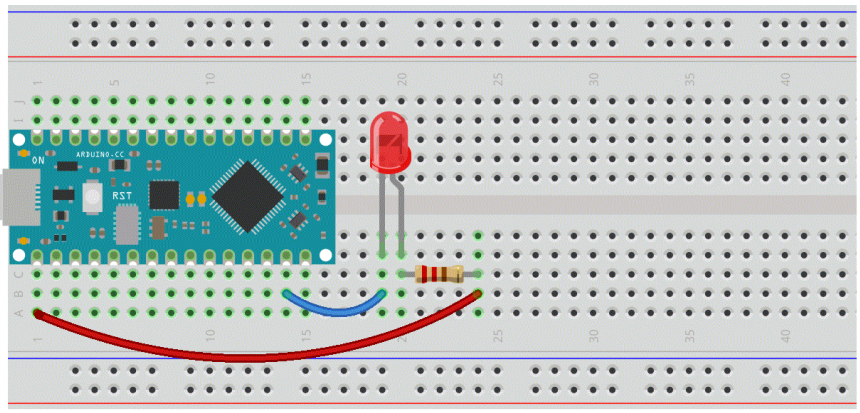
When it went fine you'll see the notifications **"Done compiling."** and **"Done uploading."** appear to let you know the process has completed correctly.

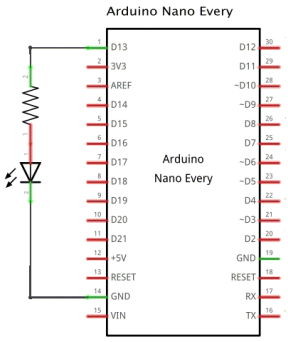
You can adjust the values of the 2 delay times to see changes in blinking rhythm. Don't forget to compile and upload the code after you made changes.

6. An External LED (for UNO)



7. An LED on a Breadboard (for Nano Every)





Then lets try some other actuators

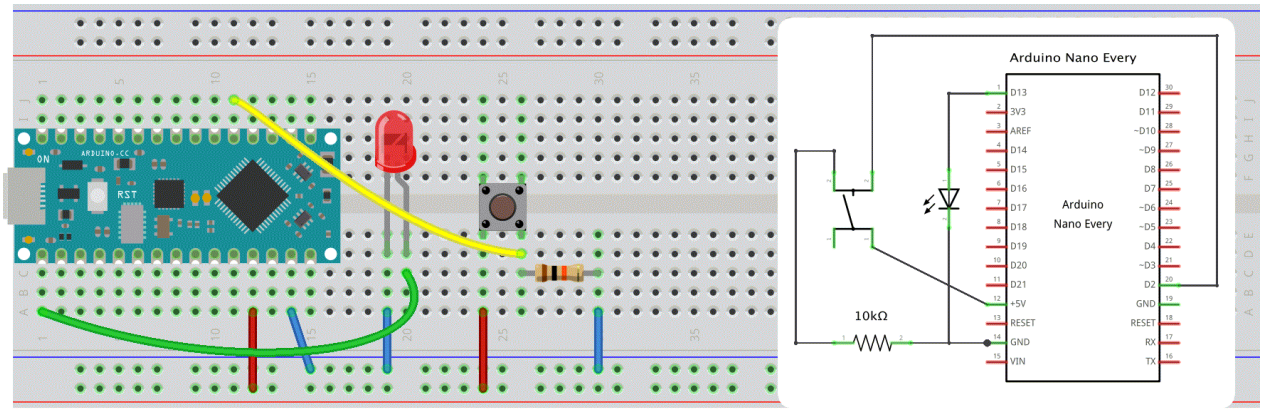
- A Buzzer or beeper is a little device that makes a buzzing noise and is used for signalling.
- A **relay** is an electrically operated switch. It uses a low voltage control signal to switch, usually higher voltage. It can also be used to control lighting, electrical and other equipment.
- …

8. Push the Button (for Nano Every)

In our first example, the LED was our actuator, and our Arduino was controlling it. If we image an outside parameter to take control over this LED, our finger, we need **a sensor**. And the simplest form of sensor available is **a pushbutton**.

Circuit

- LED attached from pin 13 to ground
- pushbutton attached to pin 2 from +5V
- 10K resistor attached to pin 2 from ground



Code

```
// constants don't change:
const int buttonPin = 2;
const int ledPin =  13;

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output
  // & the pushbutton pin as an input
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

If everything is correct, the LED will light up when you press the button.
Yes?! Good!

8b. Sticky On/Off Button (for UNO & Nano Every)

Lets program **a second behaviour** that to make the button "stick".

Code

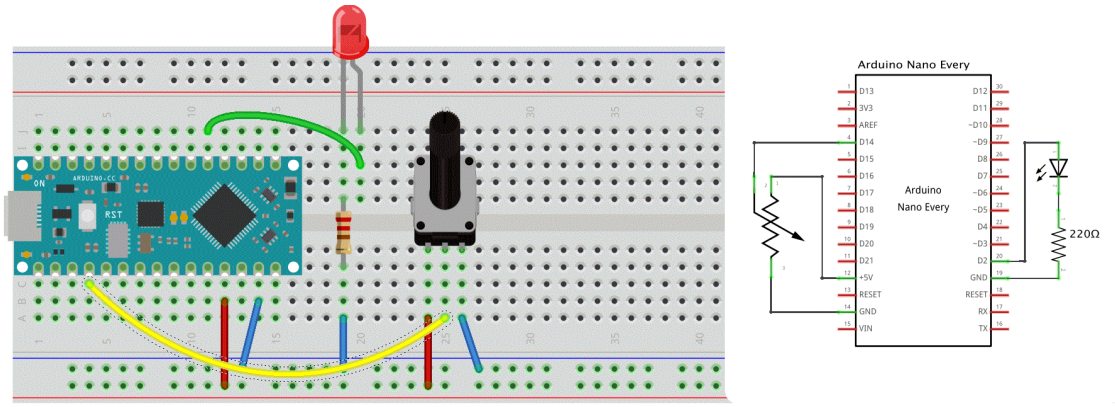
```
/* Turn on LED when the button is pressed
  and keep it on after it is released */

const int buttonPin = 2;
const int ledPin =  13;

int val = 0;          // val will be used to store the state of the input pin
int old_val = 0;      // this variable stores the previous value of "val"
int buttonState = 0;  // variable that will store the pushbutton status

void setup() {
  // initialize the LED pin as an output
  // & the pushbutton pin as an input
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  val = digitalRead(buttonPin);

  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)) {
    buttonState = 1 - buttonState;
    delay(10);    // small delay for debouncing
  }

  old_val = val;  // val is now old, let's store it

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == 1) {
    digitalWrite(ledPin, HIGH); // turn LED on
  } else {
    digitalWrite(ledPin, LOW);  // turn LED off
  }
}
```

8a. Analog Sensors (for Nano Every)

The next sketch & electronics diagram demonstrates analog input by reading an analog sensor as a potentiometer (or trimpot) on analog pin 0 and turning on and off a LED connected to digital pin 2. The amount of time the LED will be on and off depends on the value obtained by `analogRead()`.

Circuit

- potentiometer: center pin of the potentiometer to the analog input 0, one side pin (either one) to ground, the other side pin to +5V
- LED: a 220Ω resister bridges digital output 2 to the anode (long leg) of the LED, the cathode (short leg) attached to ground. Actually the resistor can also go in between the cathode and ground as in a series circuit the order of components does not matter as the current has to pass through all the parts!



Code

```
int sensorPin = A0;   // select the input pin for the potentiometer
int ledPin = 2;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT
  pinMode(ledPin, OUTPUT);
  // there is no need to set our analog in pin
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

9b. talk2me (for UNO and Arduino Every)

Lets make a **serial communication** from our Arduino to our computer to monitor our changing numbers.

In the code below we will map the 0-1023 values to a custom range 10-500, send the 2 variables over the serial port and the Arduino Serial Monitor to view them. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

The circuit remains the same.

Code

```
int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 2;        // select the pin for the LED
int sensorValue = 0;   // variable to store the value coming from the sensor
int outputValue = 0;   // variable to store a scaled value of the sensorvalue

void setup() {
  // declare the ledPin as an OUTPUT
  pinMode(ledPin, OUTPUT);
  // initialize serial communications at 9600 bps
  Serial.begin(9600);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);

  // map or scale it to a custom range:
  outputValue = map(sensorValue, 0, 1023, 10, 500);

  // print the results to the Serial Monitor:
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

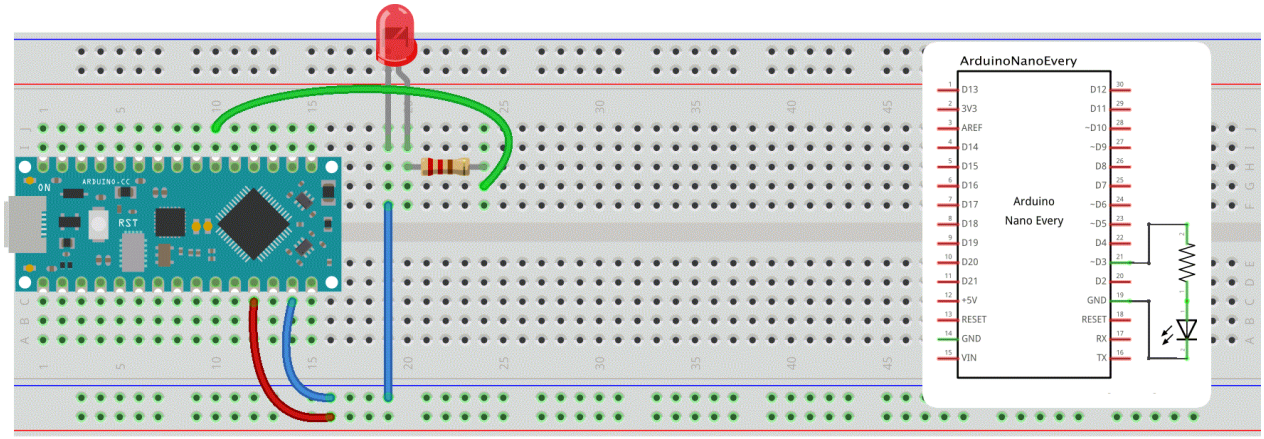
10a. Analog Out, a fading LED (for Nano Every)

PWM, short for **Pulse Width Modulation**, is a technique used to encode analog signal level into a digital one.

There are 6 PMW interfaces on an Arduino Uno: Digital pins 3, 5, 6, 9, 10, and 11, all are indicated with a ~ (tilde).

We will explore this PWM magic by changing the brightness of a LED over time.

Circuit



Code

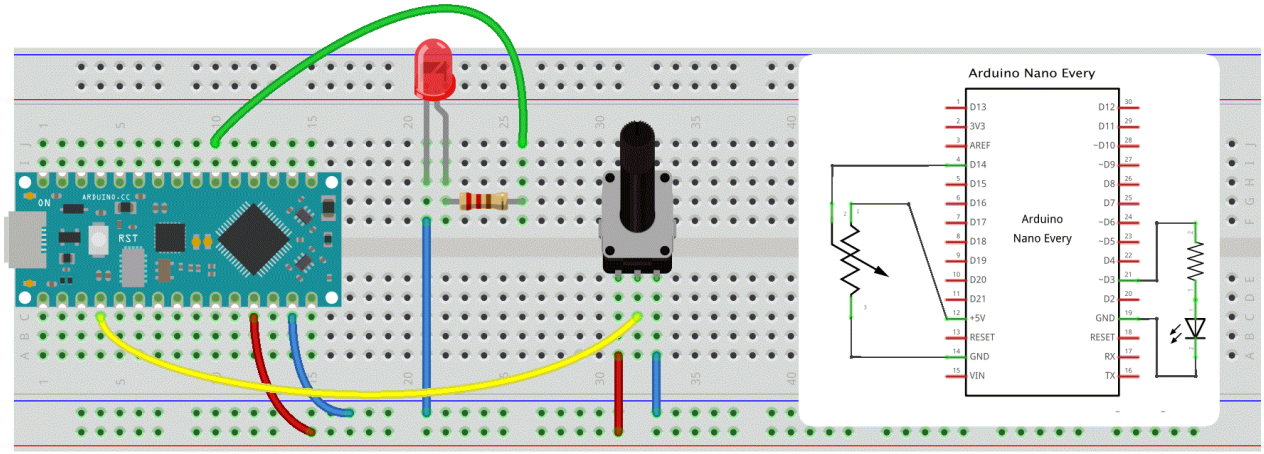
```
int ledPin = 3;      // LED connected to digital pin 3
int fadeAmount = 5;  // how many steps to fade the LED by


void setup() {
  // nothing happens in setup
}

void loop() {
  // fade in from min to max in increments of ? points:
  for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += fadeAmount) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }

  // fade out from max to min in increments of ? points:
  for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= fadeAmount) {
    // sets the value (range from 0 to 255):
    analogWrite(ledPin, fadeValue);
    // wait for 30 milliseconds to see the dimming effect
    delay(30);
  }
}
```

10b. Analog Input 2 Output (for Nano Every)

Now lets connect our Input with the Output.
In a previous experiment, we have done a *button-controlled LED*, using digital signal to control digital pin. Now we will use a potentiometer to control the brightness of the LED.

The Arduino will read the analog value of the potentiometer and assign this value to PWM port. But since PWM works only in the range from 0 - 255 we need map the input from our sensor down, thus a value between 0 - 1024 to one between 0 - 255.

Circuit



Code

```
/* Set the brightness of ledPin to a brightness specified by the
  value of the analog input */

const int ledPin = 3;      // LED connected to digital pin 9
const int analogPin = A0;  // potentiometer connected to analog pin 0

int val = 0;               // variable to store the read value
int ledVal;                // variable to store the output value


void setup() {
  // Noting here as: Analog pins are automatically set as inputs &
  // it is not needed to set the pin as an output before calling analogWrite()
}
void loop() {
  // read the value from the sensor
  val = analogRead(analogPin);
  // turn the ledpin on at the brightness set by the sensor
  //Mapping the Values between 0 to 255 because we can give output
  //from 0 -255 using the analogwrite funtion
  ledVal = map(val, 0, 1023, 0, 255);
  analogWrite(ledPin, ledVal);
  delay(10);
}
```

10c. Servo Motor Control (for Nano Every)

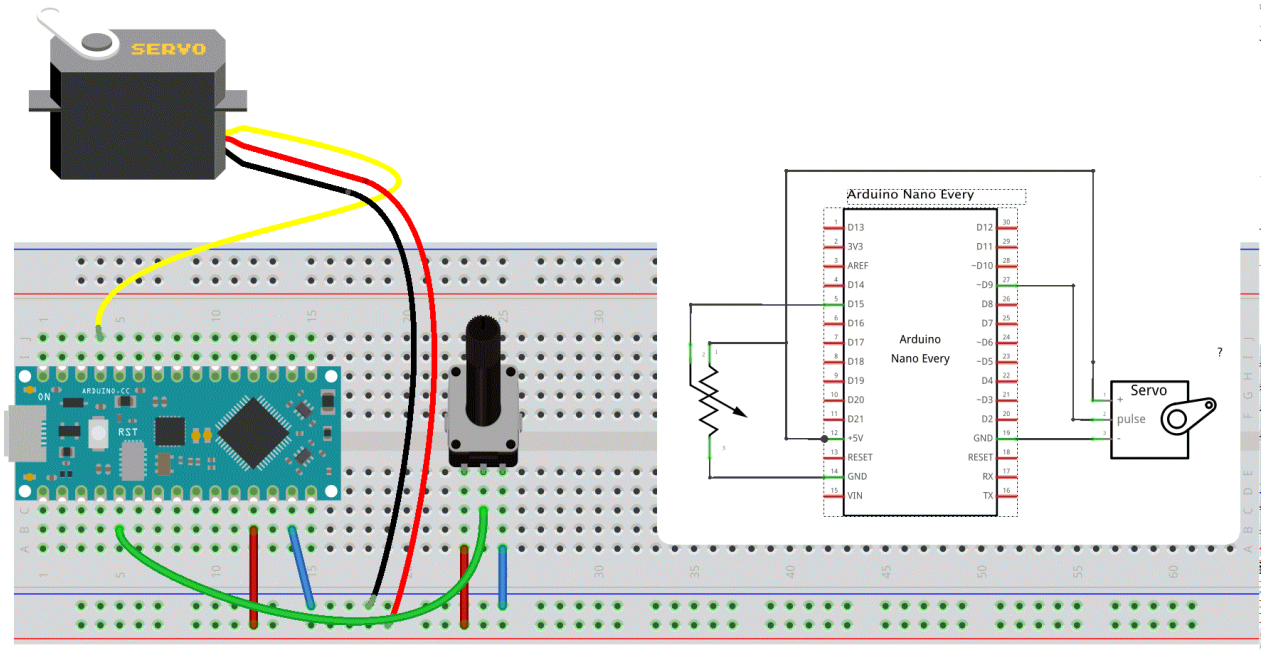
Now, lets substitute that LED for a **Servo Motor**.

Servos are motors with a shaft that can turn to a specified position. They usually have a range from 0 to 180 degrees. With an Arduino we can tell a servo to go to a specified position. We will see how to connect a servo motor and then how to turn it to different positions defined by the value of our potentiometer.

Circuit

Our servo motor has a female connector with three pins.

- The darkest, brown here, is usually the ground. Connect it to the Arduino GND.
- Connect the power cable that in all standards should be red to 5V on the Arduino.
- Connect the remaining line on the servo connector to a digital 9 (or 10) on the Arduino.



Note that servos can draw considerable power, so if you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the +5V pin on your Arduino).

Code

In this example we will use **a specific servo library** that will make coding a lot easier.

To use a library in a sketch, select it from Sketch > Import Library or just type in the `#include <name_of_library>` command.

```
#include <Servo.h>

Servo myservo;    // create servo object to control a servo

int potpin = A0;  // analog pin used to connect the potentiometer
int val;          // variable to read the value from the analog pin

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  // read the value of the potentiometer
```

```
    val = analogRead(potpin);
    // scale it to use it with the servo (value between 0 and 180)
    val = map(val, 0, 1023, 0, 180);
    // sets the servo position according to the scaled value
    myservo.write(val);
    // waits for the servo to get there
    delay(15);
  }
```

See the servo reference page on how to use it.

The key functions used here are:

- Servo *objectname*;
- *objectname*.attach*(interface)* select the pin for servo. This can only use pin 9 or 10.
- *objectname*.write*(angle)* used to control the angle of the servo (0 to 180 degree).