Introduction to Arduino

Material in this document has been compiled from various tutorials available on the net for Arduino.

1 Arduino IDE Installation

Installation of the Integrated Development Environment (IDE) for Arduino requires i) downloading the software and ii) installing it using the installation utilities of your operating system.

1.1 Downloading Arduino IDE 2.1

IDE software can be downloaded from the Arduino Software page:
`https://www.arduino.cc/en/software#future-version-of-the-arduino-ide`

Operating System version requirements for running the IDE are:

Windows: Win 10 and newer, 64 bits, or

Mac OS X: Version 10.14: "Mojave" or newer, 64 bits, or

Linux: 64 bit distributions

Arduino IDE 2.1 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped User Interface or UI, improved board and library manager, debugger, auto-complete feature and much more.

To download the Editor, you need to visit the Arduino software page. Go to:
`https://docs.arduino.cc/software/ide-v2` and click on the Download button.

1.2 Installing IED 2.0

Installation procedure is operating system dependent.

Windows: To install Arduino IDE 2.1 on a Windows computer, simply run the file downloaded from the software page. Follow the instructions in the installation guide. The installation may take several minutes.

MAC OS: To install the Arduino IDE 2.1 on a MAC OS computer, simply copy the downloaded file into your application folder.

Linux: To install Arduino IDE 2.1 on Linux, first download the file:
`arduino-ide_2.1.0_Linux_64bit.zip`
from the software page and place it in a directory of your choice. The file is compressed and you have to extract it in a suitable folder, remembering that it will be executed from there.

For this, open a terminal window in this directory and unzip the file:
`sudo unzip arduino-ide_2.1.0_Linux_64bit.zip`

This will extract all the necessary files from the zip archive. You will find an executable file "arduino-ide" in the directory where you unzipped the downloaded file. Running arduino-ide from this directory (for example, by double clicking on the file name) will start the IDE program.

If you have a bin directory added to your path, you can place a link to this executable file in your bin directory.

```
cd <path to your bin directory>
ln -s arduino-ide <arduino directory path>/arduino-ide
```

Now you will be able to launch the IDE from any directory. This step is optional.

Arduino programs are called "sketches". These sketches are compiled on the host PC and uploaded using a serial connection to the Arduino board. Linux is a multi-user OS. Access to common resources like the serial or USB port is controlled by the OS – otherwise multiple users may use a resource simultaneously and destroy each others' data. Hence, access to common resources has to be obtained from the system administrator, before you can use these.

When you start the IDE on your PC, you have to select a board (so that the "sketch" will be compiled according to specifics of your board) and a port, which will be used to communicate with your board.

It is possible that when you upload a sketch - (after selecting your board and the serial port), - you get an error:
"Error opening serial port ...", or
"can't open device "/dev/ttyUSB0": Permission denied
If you get this error, you need to set serial port permissions.

Open a terminal and run ls -l on the problem device. For this, type:
ls -l /dev/ttyACM*
or
ls -l /dev/ttyUSB*

you will get something like:

crw-rw—- 1 root dialout 188, 0 5 apr 23.01 ttyACM0
or
crw-rw—-. 1 root dialout 188, 0 Nov 8 11:44 /dev/ttyUSB0

The "0" at the end of ACM/USB might be a different number depending on the actual serial line of USB port being used, or multiple entries might be returned. We want to find the group owner of the tty – in this case, it is "dialout".

Now we just need to add our user to the group: This is done by:

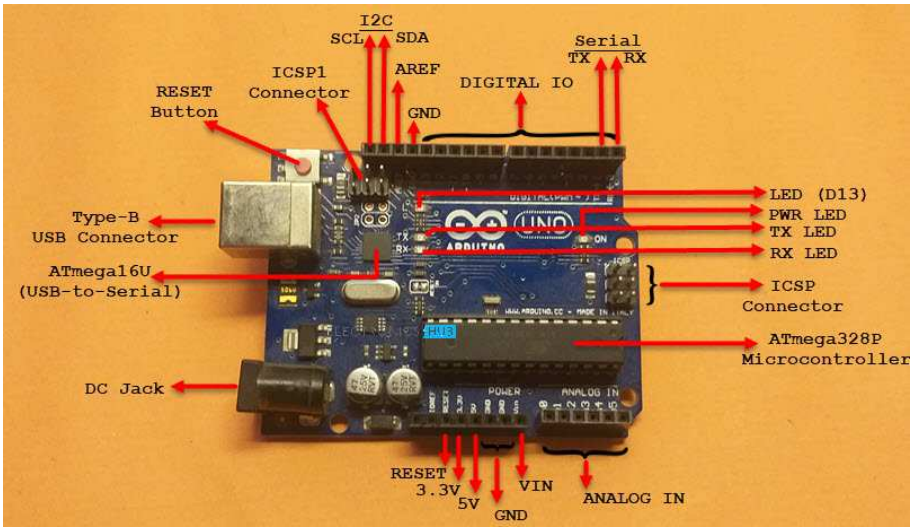
sudo usermod -a -G dialout `<username>`

where `<username>` is your Linux user name. This will add you to the group "dialout" which owns this serial line. You will need to log out and log in again for this change to

take effect.

After this procedure, you should be able to proceed normally and upload the sketch to your board or use the Serial Monitor.

2 Arduino board pinout



The picture on the left shows an Arduino UNO board.

Pins are brought out to the connectors on the side of the board and pin names are silk-screened on the board next to the pins.

Pinout for Arduino Uno

Pin Category	Pin Name	Details
Power:		
	Vin	Input voltage to Arduino when using an external power source.
	5V	Regulated power supply used to power microcontroller, other components
	3.3V	3.3V supply generated by on-board voltage regulator. Max current: 50mA.
	GND	ground pins.
I/O Pins:		
Digital Pins	0 - 13	Digital input or output pins
Pulse Width Modulation	3, 5, 6, 9, 11	8-bit PWM output.
Analog Pins	A0 – A5	Used to provide analog input in the range of 0-5V
Analog Ref	AREF	To provide reference voltage for input voltage.
Serial	0(Rx), 1(Tx)	Receive and Transmit TTL serial data.
SPI	11 (MOSI), 12 (MISO) 10 (SS), 13 (SCLK)	Used for SPI communication
I2C interface	A4 (SDA), A5 (SCA)	for “two wire interface” (TWI) or I2C communication
External Interrupts	2, 3	To trigger an interrupt
Reset	Reset	Resets the microcontroller.
Inbuilt LED	13	To turn on the inbuilt LED

3 Language Reference

Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

3.1 functions

For controlling the Arduino board and performing computations:
Digital I/O: digitalRead(), digitalWrite(), pinMode()
Analog I/O: analogRead(), analogReference() analogWrite()

on Zero, Due and MKR Family: analogReadResolution(), analogWriteResolution()
Advanced I/O: noTone(), tone()
pulseIn(), pulseInLong(), shiftIn(), shiftOut()
Time, delay(), delayMicroseconds(), micros(), millis(),

Math:
abs(), constrain(), map(), max(), min(), pow(), sq(), sqrt() Trigonometry:
cos(), sin(), tan()
Characters:
isAlpha(), isAlphaNumeric(), isAscii(), isControl(), isDigit(), isGraph()
isHexadecimalDigit(), isLowerCase(), isPrintable(), isPunct(), isSpace()
isUpperCase(), isWhitespace()
Random Numbers:
random(), randomSeed()
Bits and Bytes:
bit(), bitClear(), bitRead(), bitSet(), bitWrite()
highByte(), lowByte()
Interrupt controls:
interrupts(), noInterrupts()
External Interrupts:
attachInterrupt(), detachInterrupt()
Communication:
Serial, SPI, Stream, Wire, USB, Keyboard, Mouse

3.1.1 Variables:

Arduino data types and constants:
Constants:
HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTIN
true | false
Floating Point Constants, Integer Constants
Conversion:
(unsigned int), (unsigned long)
byte(), char(), float(), int(), long(), word()

3.1.2 Data Types:

array, bool, boolean, byte, char, double, float, int, long, short,
size_t, string, String(), unsigned char

unsigned int, unsigned long, void, word,
Variable Scope and Qualifiers:
const, scope, static, volatile, Utilities, PROGMEM, sizeof().
Structure:
The elements of Arduino (C++) code: Sketch, loop(), setup()
Control Structure: break, continue, do...while, else, for, goto, if
return, switch...case, while,
Further Syntax:
#define (define), #include (include) `/* */` (block comment), `//` (single line comment)
; (semicolon), {} (curly braces)
Arithmetic Operators:
% (remainder), * (multiplication), + (addition), - (subtraction), / (division)
= (assignment operator)
Comparison Operators:
== (equal to), != (not equal to), < (less than), <= (less than or equal to)
> (greater than), >= (greater than or equal to)
Boolean Operators:
! (logical not), && (logical and), || (logical or)
Pointer Access Operators:
& (reference operator), * (dereference operator)
Bitwise Operators:
& (bitwise and), `<<` (bitshift left), `>>` (bitshift right),
ˆ (bitwise xor), | (bitwise or), ˜ (bitwise not)
Compound Operators:
%= (compound remainder), &= (compound bitwise and)
= (compound multiplication), ++ (increment), += (compound addition)
– (decrement), -= (compound subtraction), /= (compound division)
ˆ= (compound bitwise xor), | = (compound bitwise or)

4 Example code

Sketches described in this section are available from the File/Examples menu on the IDE.

4.1 Bare Minimum code example

The bare minimum of code needed to start an Arduino sketch:

This example contains the bare minimum of code you need for a sketch to compile properly on Arduino Software (IDE): the setup() method and the loop() method.

4.1.1 Hardware Required

Arduino Board

4.1.2 Circuit

Only your Arduino Board is needed for this example.

4.1.3 Code

The bare minimum of code needed to start an Arduino sketch:

This example contains the bare minimum of code you need for a sketch to compile properly on Arduino Software (IDE):
the setup() method and the loop() method.

This code won't actually do anything, but it's structure is useful for copying and pasting to get you started on any sketch of your own. It also shows you how to make comments in your code.

```
void setup () {
// put your set up code here to run once:
}

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

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the board.

After creating a setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond as it runs. Code in the loop() section of your sketch is used to actively control the board.

Any line that starts with two slashes (//) will not be read by the compiler, so you can write anything you want after it. The two slashes may be put after functional code to keep comments on the same line. Commenting your code like this can be particularly helpful in explaining, both to yourself and others, how your program functions step by step.

4.2 Blinking an LED:

This example shows the simplest thing you can do with an Arduino to see physical output: it blinks the on-board LED.

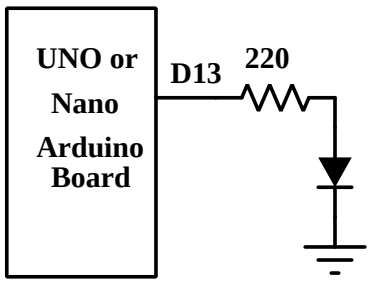
4.2.1 Hardware Required

Arduino Board, and (optional) LED, 220 Ω resistor

4.2.2 Circuit

This example uses the built-in LED that most Arduino boards have. The LED is connected to a digital pin and the pin number may vary from board type to board type. To make your life easier, we have a constant that is specified in every board descriptor file. This constant is `LED_BUILTIN` and allows you to control the built-in LED easily. Here is the correspondence between the constant and the digital pin:
Boards 101 Due, Intel Edison, Intel Galileo Gen2, Leonardo and Micro, LilyPad, LilyPad USB, MEGA2560, Mini, Nano, Pro, Pro Mini, UNO, Yún and Zero connect D13 to the LED. Gemma board connects the LED to D1 and MKR1000 connects it to D6.

If you want to light an external LED with this sketch, you need to build this circuit, where you connect one end of the resistor to the digital pin correspondent to the `LED_BUILTIN` constant. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the short leg of the LED (the negative leg, called the cathode) to the GND. In the diagram below we show an UNO board that has D13 as the `LED_BUILTIN` value.



The value of the resistor in series with the LED may be of a different value than 220 Ω; the LED will light up also with values up to 1K Ω.

4.2.3 Code

```
/*
  Blink Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products
*/

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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);// turn the LED on (HIGH is the voltage level)
  delay(1000);                    // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000);                    // wait for a second
}
```

4.3 Digital Read Serial

Read a switch, print the state out to the Arduino Serial Monitor.

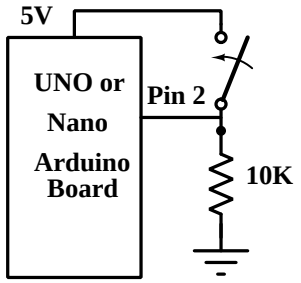
This example shows you how to monitor the state of a switch by establishing serial communication between your Arduino and your computer over USB.

4.3.1 Hardware Required

Arduino Board, a momentary switch, button, or toggle switch, 10K Ω resistor, hook-up wires and breadboard.

4.3.2 Circuit

When the push button is open (unpressed) there is no connection between the two legs of the push button, so the pin is connected to ground (through the 10KΩ pull-down resistor) and reads as LOW, or '0'. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or '1'.



4.3.3 code

In the program below, the very first thing that you do (in the setup function) is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the line:

Serial.begin(9600);

Next, initialize digital pin 2, the pin that will read the output from your button, as an input:

pinMode(2,INPUT);

Now that your setup has been completed, move into the main loop of your code. When your button is pressed, 5 volts will freely flow to pin 2 and when it is not pressed, pin 2 will be connected to ground through the 10 K Ω resistor. This is a digital input, meaning that the switch can only be in either an on state (seen by your Arduino as a '1', or HIGH) or an off state (seen by your Arduino as a '0', or LOW), with nothing in between.

The first thing you need to do in the main loop of your program is to establish a variable to hold the information coming in from your switch. Since the information coming in from the switch will be either a '1' or a '0', you can use an int datatype. Call this variable 'sensorValue', and set it to equal whatever is being read on digital pin 2. You can accomplish this with just one line of code:

int sensorValue = digitalRead(2);

Once the board has read the input, make it print this information back to the computer as a decimal value. You can do this with the command Serial.println() in our last line of code:

Serial.println(sensorValue);

Now, when you open your Serial Monitor in the Arduino Software (IDE), you will see a stream of '0's if your switch is open, or '1's if your switch is closed.

```
/*
  DigitalReadSerial
  Reads a digital input on pin 2, prints the result to the Serial Monitor
*/

// digital pin 2 has a push button attached to it. Give it a name:
int pushButton = 2;
```

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
```

4.4 Fading a LED

This example demonstrates the use of analog output to fade an LED. We use of the analogWrite() function to fade-in and fade-out an LED.

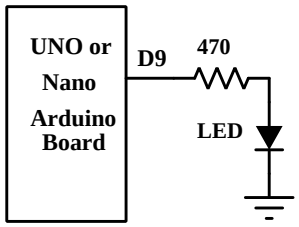
AnalogWrite uses pulse width modulation (PWM). In PWM, we turn something on and off very quickly, controlling the ratio of on and off durations. The ratio of the two durations determines the average value at the output. Thus we can generate an average value on the output pin which is not the supply voltage or ground, but can be adjusted to any value in between. Hence it is called "analogWrite". We can use this technique to create a fading effect.

4.4.1 Hardware Required

Arduino board, LED, 220Ω resistor, hook-up wires, breadboard,

4.4.2 Circuit

Connect the anode (the longer, positive leg) of your LED to digital output pin 9 on your board through a 220 Ω resistor. Connect the cathode (the shorter, negative leg) directly to ground.



4.4.3 Code

After declaring pin 9 to be your LED Pin, there is nothing to do in the setup() function of your code. The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what PWM value to write.

In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the sketch below, the PWM value is set using a variable called 'brightness'. Each time through the loop, it increases by the value of the variable 'fadeAmount'. If brightness is at either extreme of its value (either 0 or 255), then fadeAmount is changed to its negative. In other words, if

fadeAmount is 5, then it is set to -5. If it's -5, then it's set to 5. The next time through the loop, this change causes brightness to change direction as well.

The function analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the fading effect.

```
/*
  Fade

  This example shows how to fade an LED on pin 9 using the analogWrite()
  function.
  The analogWrite() function uses PWM, so if you want to change the pin
  you're using, be sure to use another PWM capable pin. On most Arduinos,
  the PWM pins are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10
  and ~11.
*/

int led = 9;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

4.5 Reading an analog voltage

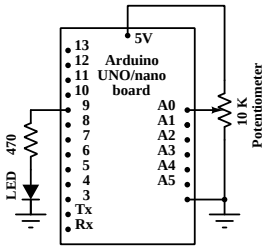
This example shows you how to read an analog input on analog pin 0, convert it into voltage, and print it out to the serial monitor of the Arduino Software (IDE).

4.5.1 Hardware Required

Arduino Board, 10KΩ potentiometer.

4.5.2 Circuit

Connect three wires from the potentiometer to your board. The first goes to ground from one of the outer pins of the potentiometer. The second goes to 5 volts from the other outer pin of the potentiometer. The third goes from the middle pin of the potentiometer to analog input 0. By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin, which will be read using an analog to digital converter.



When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 KΩ), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the analog voltage that you're reading as an input.

The microcontroller of the board has a circuit inside called an analog-to-digital converter or ADC that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

4.5.3 Code

In the program below, the very first thing that you do will in the setup function is to begin serial communications, at 9600 bits of data per second, between your board and your computer with the line:
Serial.begin(9600);
Next, in the main loop of your code, you need to establish a variable to store the resistance value (which will be between 0 and 1023, perfect for an int datatype) coming in from your potentiometer:
int sensorValue = analogRead(A0);
To change the values from 0-1023 to a range that corresponds to the voltage the pin is reading, you'll need to create another variable, a float, and do a little math. To scale the numbers between 0.0 and 5.0, divide 5.0 by 1023.0 and multiply that by sensorValue :
float voltage= sensorValue * (5.0 / 1023.0);
Finally, you need to print this information to your serial window. You can do this with the command Serial.println() in your last line of code:
Serial.println(voltage)
Now, when you open your Serial Monitor in the Arduino IDE (by clicking on the icon on the right side of the top green bar or pressing Ctrl+Shift+M), you should see a steady stream of numbers ranging from 0.0 - 5.0. As you turn the pot, the values will change, corresponding to the voltage coming into pin A0.

```
/*
  ReadAnalogVoltage
  Reads an analog input on pin 0, converts it to voltage,
  and prints the result to the Serial Monitor.
  Graphical representation is available using Serial Plotter
  (Tools > Serial Plotter menu).
  Attach the center pin of a potentiometer to pin A0, and
```

```
  the outside pins to +5V and ground.
*/
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
// read the input on analog pin 0:
  int sensorValue = analogRead(A0);
//Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
// print out the value you read:
  Serial.println(voltage);
}
```

4.6 Reading the timer

Sometimes you need to do two things at once. For example you might want to blink an LED while reading a button press. In this case, you can't use delay(), because Arduino pauses your program during the delay(). If the button is pressed while Arduino is paused waiting for the delay() to pass, your program will miss the button press.

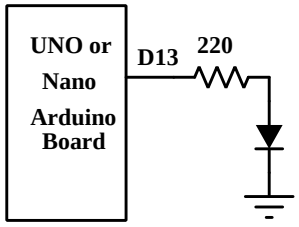
This sketch demonstrates how to blink an LED without using delay(). It turns the LED on and then makes note of the time. Then, each time through loop(), it checks to see if the desired blink time has passed. If it has, it toggles the LED on or off and makes note of the new time. In this way the LED blinks continuously while the sketch execution never lags on a single instruction.

4.6.1 Hardware Required

Arduino Board, LED, 220 Ω resistor.

4.6.2 Circuit

To build the circuit, connect one end of the resistor to pin 13 of the board. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the short leg of the LED (the negative leg, called the cathode) to the board GND, as shown in the schematic. Or else, you can just use the inbuilt LED on the board.



4.6.3 Code

The code uses the millis() function, a command that returns the number of milliseconds since the board started running its current sketch, to blink an LED.

```
/*
  Blink without Delay
```

```
  Turns on and off a light emitting diode (LED) connected to a digital pin,
  without using the delay() function. This means that other code can run
  at the same time without being interrupted by the LED code.

  The circuit:
  - Use the onboard LED.
  - Note: Most Arduinos have an on-board LED you can control. On the UNO, MEGA
    and ZERO it is attached to digital pin 13, on MKR1000 on pin 6.
    LED_BUILTIN is set to the correct LED pin independent of which board is
    used. If you want to know what pin the on-board LED is connected to on
    your Arduino model, check the Technical Specs of your board at:
    https://www.arduino.cc/en/Main/Products
*/

// constants won't change. Used here to set a pin number:
const int ledPin =  LED_BUILTIN;// the number of the LED pin

// Variables will change:
int ledState = LOW;             // ledState used to set the LED

// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;// will store last time LED was updated

// constants won't change:
const long interval = 1000;      // interval at which to blink (milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.

  // check to see if it's time to blink the LED; that is, if the difference
  // between the current time and last time you blinked the LED is bigger than
  // the interval at which you want to blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }

    // set the LED with the ledState of the variable:
```

```
    digitalWrite(ledPin, ledState);
  }
}
```

The values of currentMillis and previousMillis will keep increasing here and eventually overflow. By using unsigned long type, we can delay the overflow, but ideally, we should reset the timer periodically.

(Actually, this is a bad program from the internet – one should not write code in which some variable just overflows! Write a better version in which variables are reset rather than allowed to overflow.)