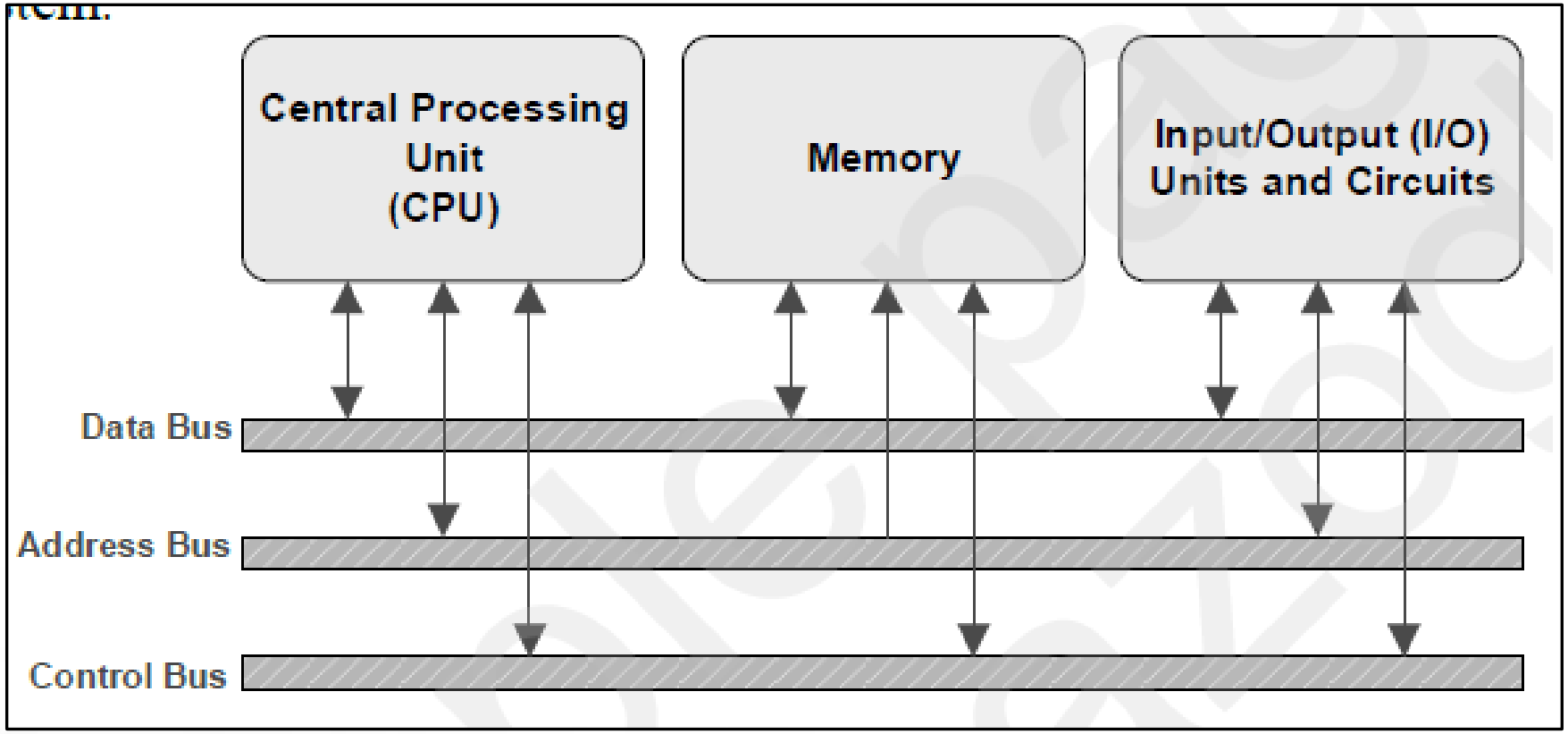
Microcontroller Basics Structure

Review – Computer Hardware

A Typical Computer Structure

A computer system (Computer-Personal Computer-PC) consists of various components (internal and external) in order to be exploited by the users that select the corresponding applications. A typical computer structure is shown below.



Basic Computer Components

Basic computer components are

1) Central processing unit (CPU): The CPU reads and executes the instructions (programs/codes) for producing the corresponding result. The CPU is implemented in an integrated circuit (IC-chip) which is called *microprocessor*.
2) Memory: The memory stores the instructions (programs/codes) and data. The memory constitutes an one dimensional array with identical locations in terms of capacity. On the other hand, every memory location has a unique identifier which is called address. For reading or writing to a memory location the corresponding address has to be activated.
3) Input/Output (I/O) units: The I/O units ensure the system functionality and the communication (read/write) with external devices to other computers/users. A well-known example is the keyboard for entering data to the computer.

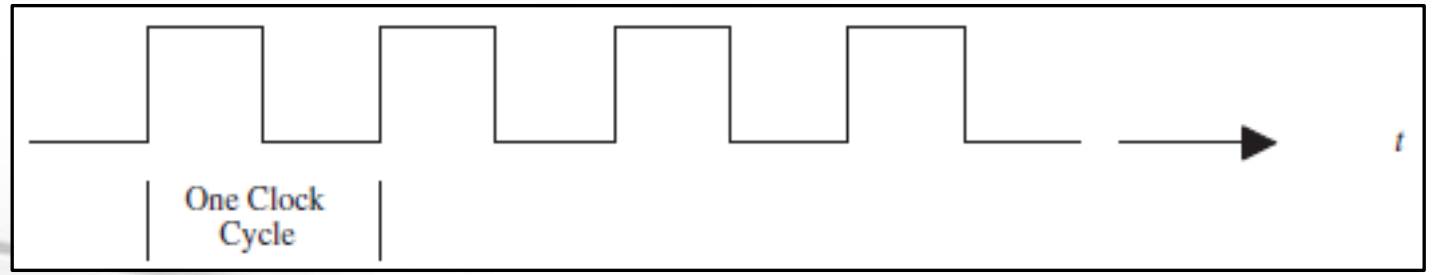
Basic Computer Components

4) System bus: It consists of data bus, address bus, and control bus.
   a) *Data bus*: The data bus is common among the computer components in order to control and exchange data between the computer components. For example, the result of the mentioned addition is transferred to the memory via the data bus.
   b) *Address bus*: Addresses are unique numbers that are assigned to all computer components for supporting the corresponding communication. For example, the result of an addition be stored in the memory, whose location is identified by the address bus.

Basic Computer Components

c) *Control bus*: The data exchange between computer components is achieved via special control signals that are transferred through the control bus.

- For example, when a computer want to write a data to a memory whose location is specified in the address bus, the control bus could identify the write signal.
- Clock signals: The system clock signals are contained in the control bus. These signals generate the appropriate clock periods during which instruction executions are carried out by the CPU. The number of cycles per second (Hertz, abbreviated as Hz) is referred to as the clock frequency. The CPU clock frequencies of typical microcontrollers vary from 1MHz to 40MHz. The execution times of microcontroller instructions are provided in terms of the number of clock cycles.

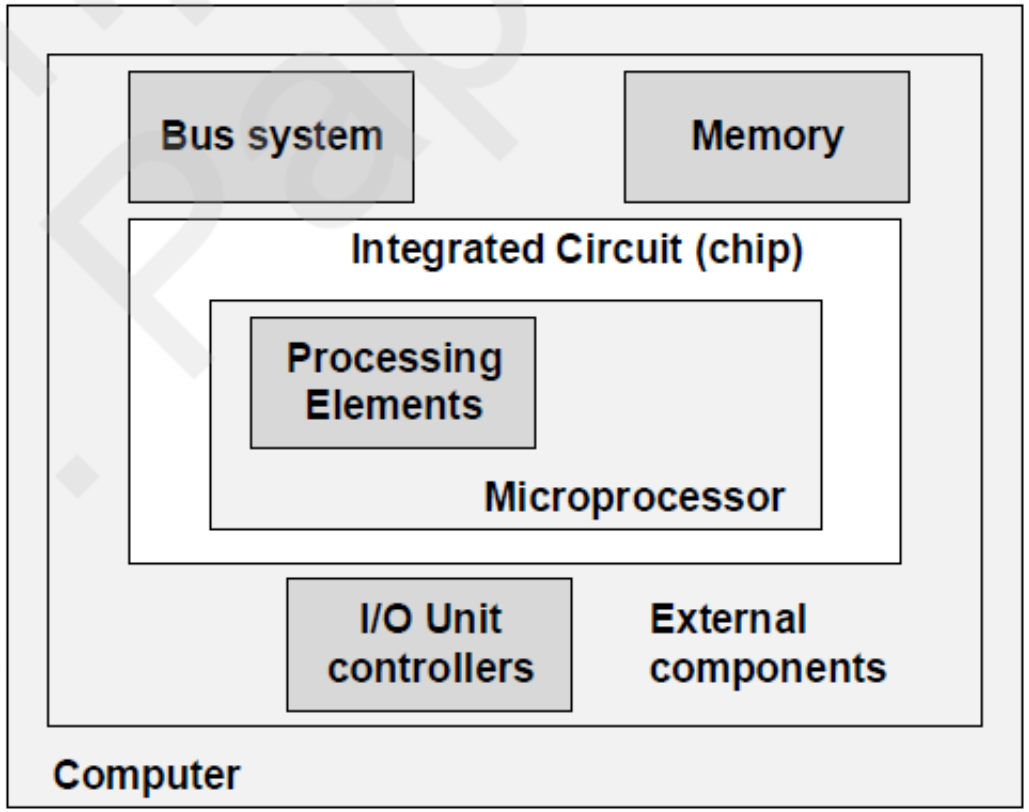


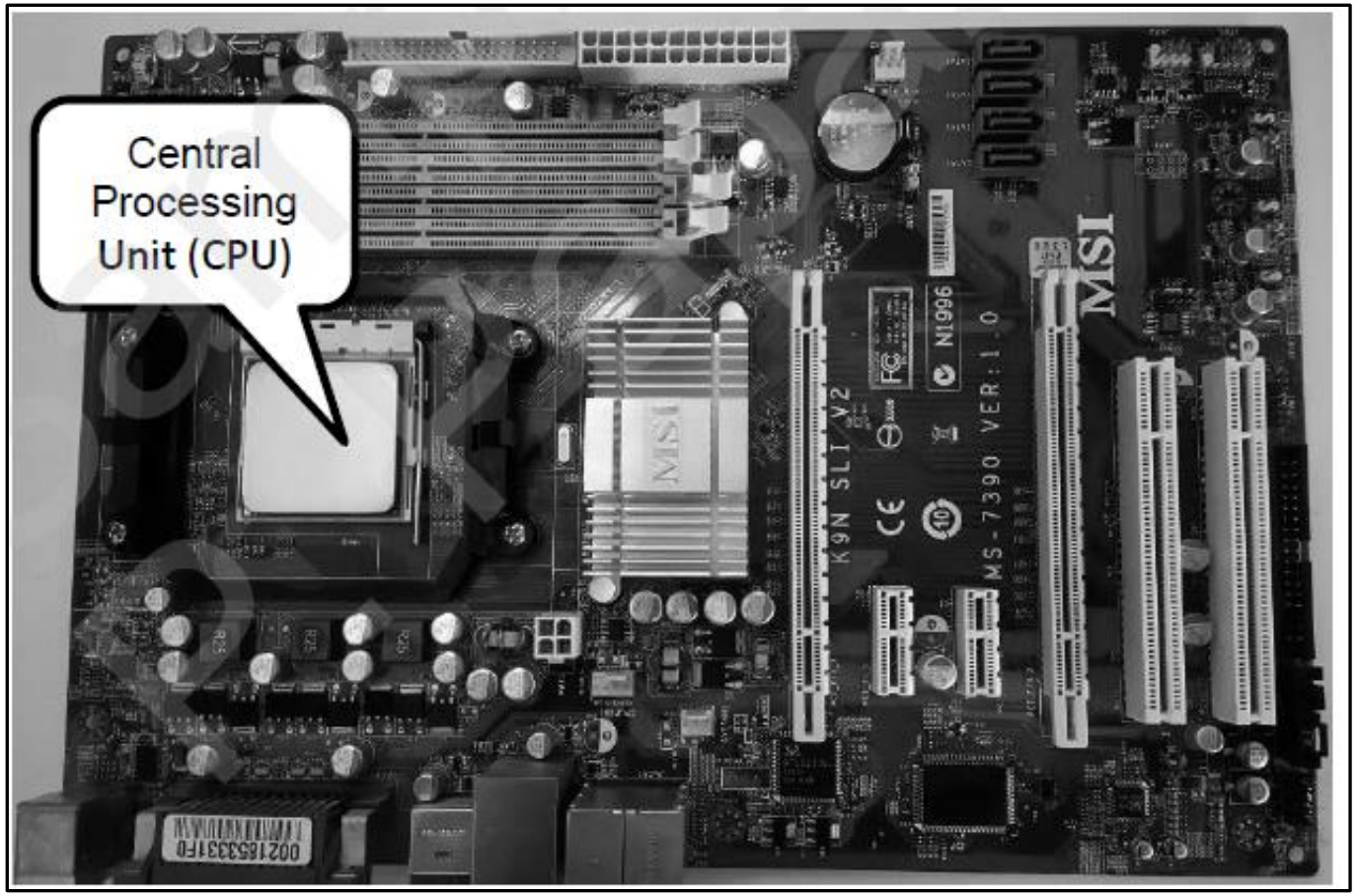
Microprocessor, Microcomputer, and Microcontroller

The differences among microprocessor, microcomputer, and microcontroller are explained as follows.

- Microprocessor: It is a central processing unit (CPU) whose components -- arithmetic logic unit (ALU), instruction decoder, registers, bus control circuit, etc. -- on a single chip.
- Microcomputer: When a microprocessor and associated support circuitry, peripheral I/O components and memory (program as well as data) were put together to form a small computer specifically for data acquisition and control applications, it was called a microcomputer.

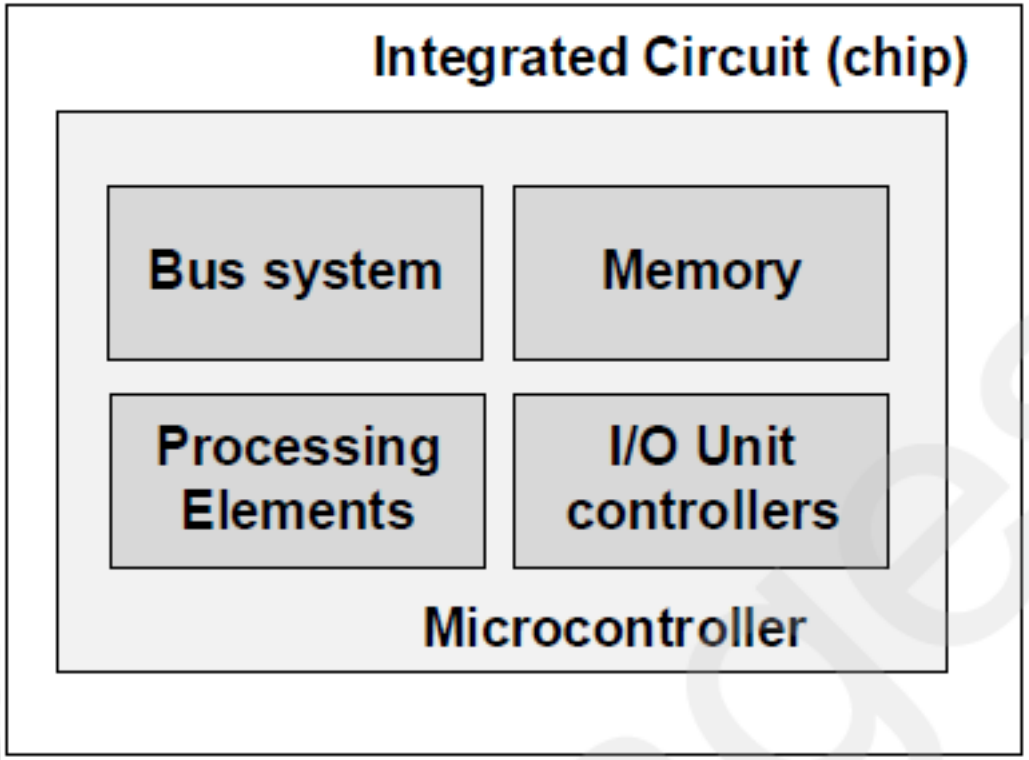
Microprocessor, Microcomputer, and Microcontroller

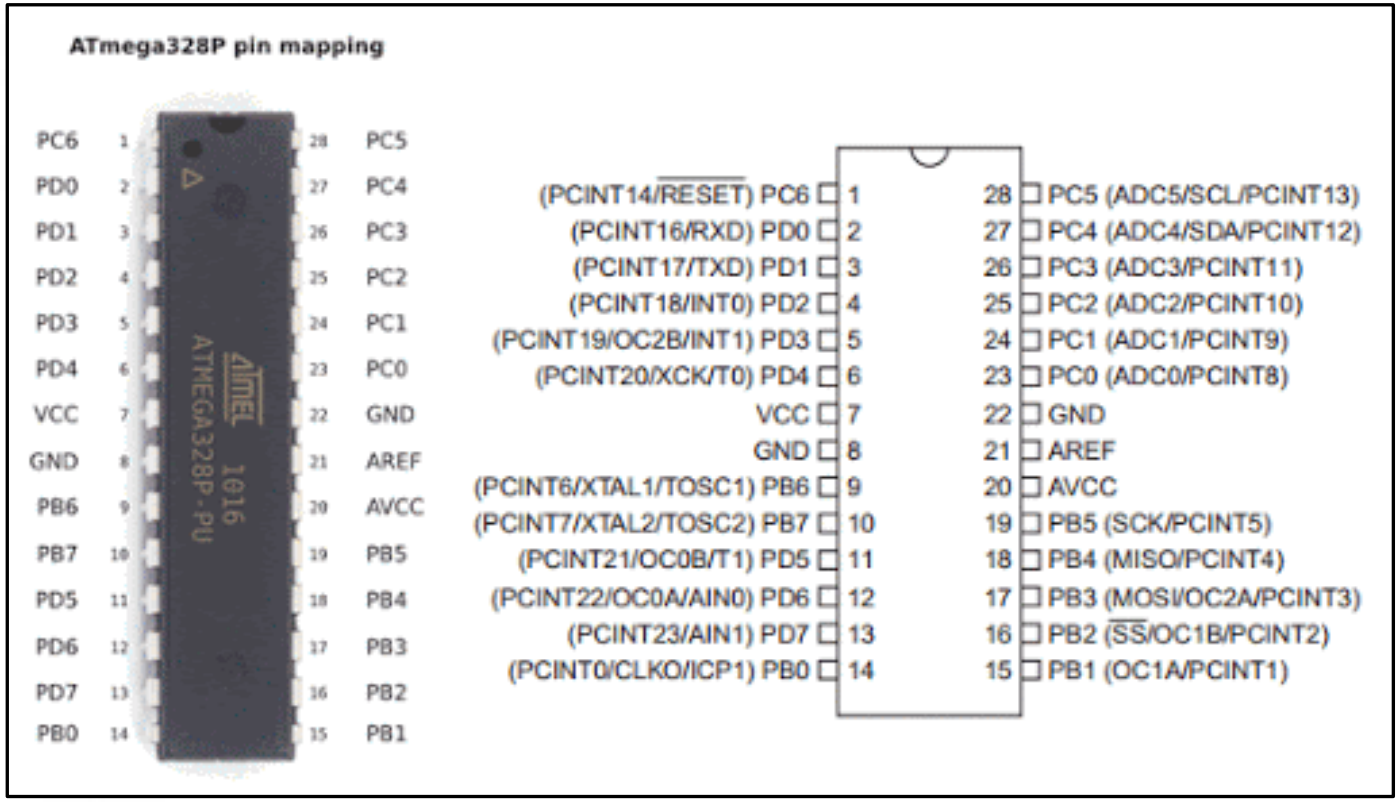




Microprocessor, Microcomputer, and Microcontroller

- Microcontroller: When the components that make a microcomputer are put together on a single chip of silicon, it is called the microcontroller. Therefore, a microcontroller consists of CPU, memory, I/O units, bus system, and pins.





Types of Microcontroller Architectures

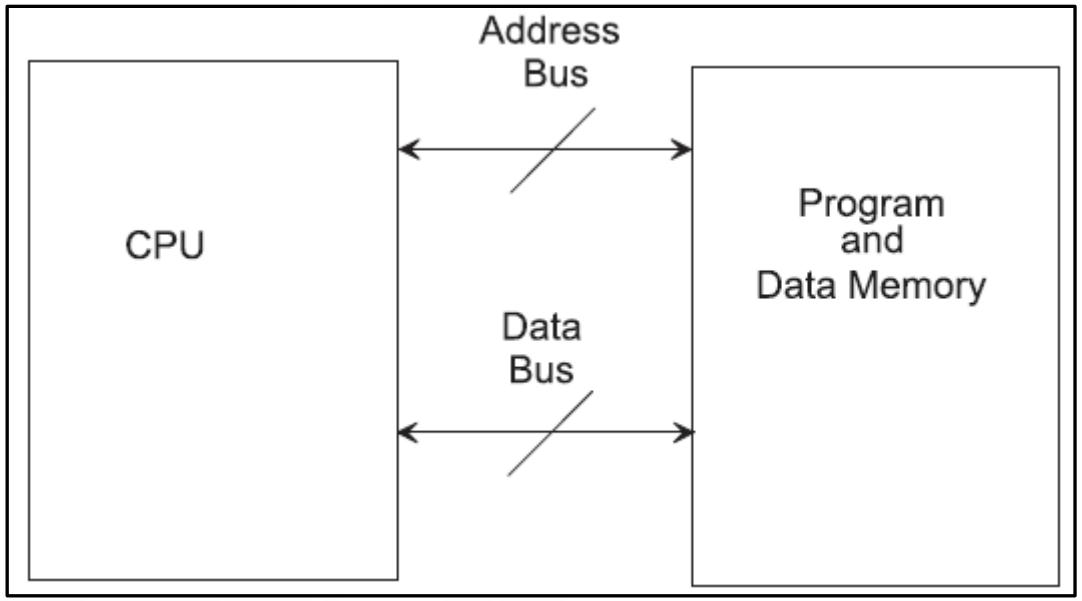
The microcontroller requires memory to store programs and data; respectively, called *program memory* and *data memory*.

- *Program memory* contains instructions and immediate data (data included with instructions).
- *Data memory* contain data in processing.

Types of Microcontroller Architectures

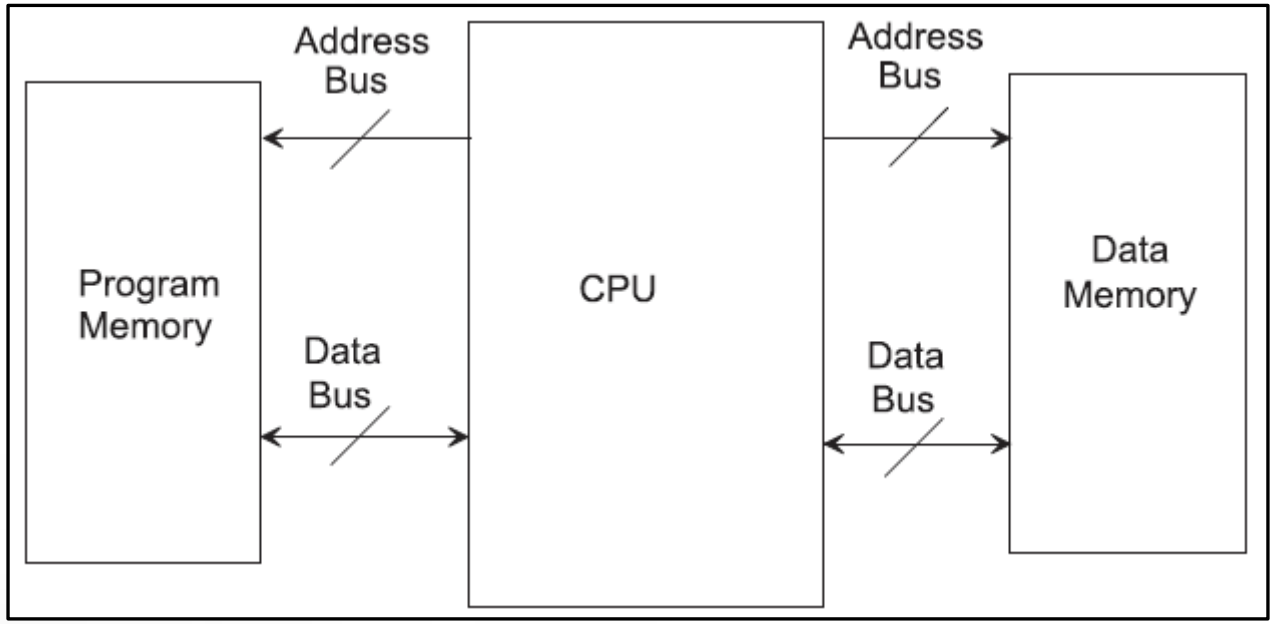
Two types of architectures are used for designing microcontrollers. They are *von Neumann (Princeton) architecture* and *Harvard architecture*. The main difference between the two architectures is the way programs and data are stored in memory.

- In von Neumann architecture, a *single memory system with the same address and data buses* is used for accessing both programs and data. This means that programs and data cannot be accessed simultaneously. This may slow down the overall speed.



Types of Microcontroller Architectures

- Harvard architecture is a type of architecture which uses *separate program and data memory units along with separate buses for instructions and data*. This means that these processors can execute instructions and access data simultaneously. Processors designed with this architecture require four buses for program memory and data memory: one data bus for instructions, one address bus for addresses of instructions, one data bus for data, and one address bus for addresses of data.



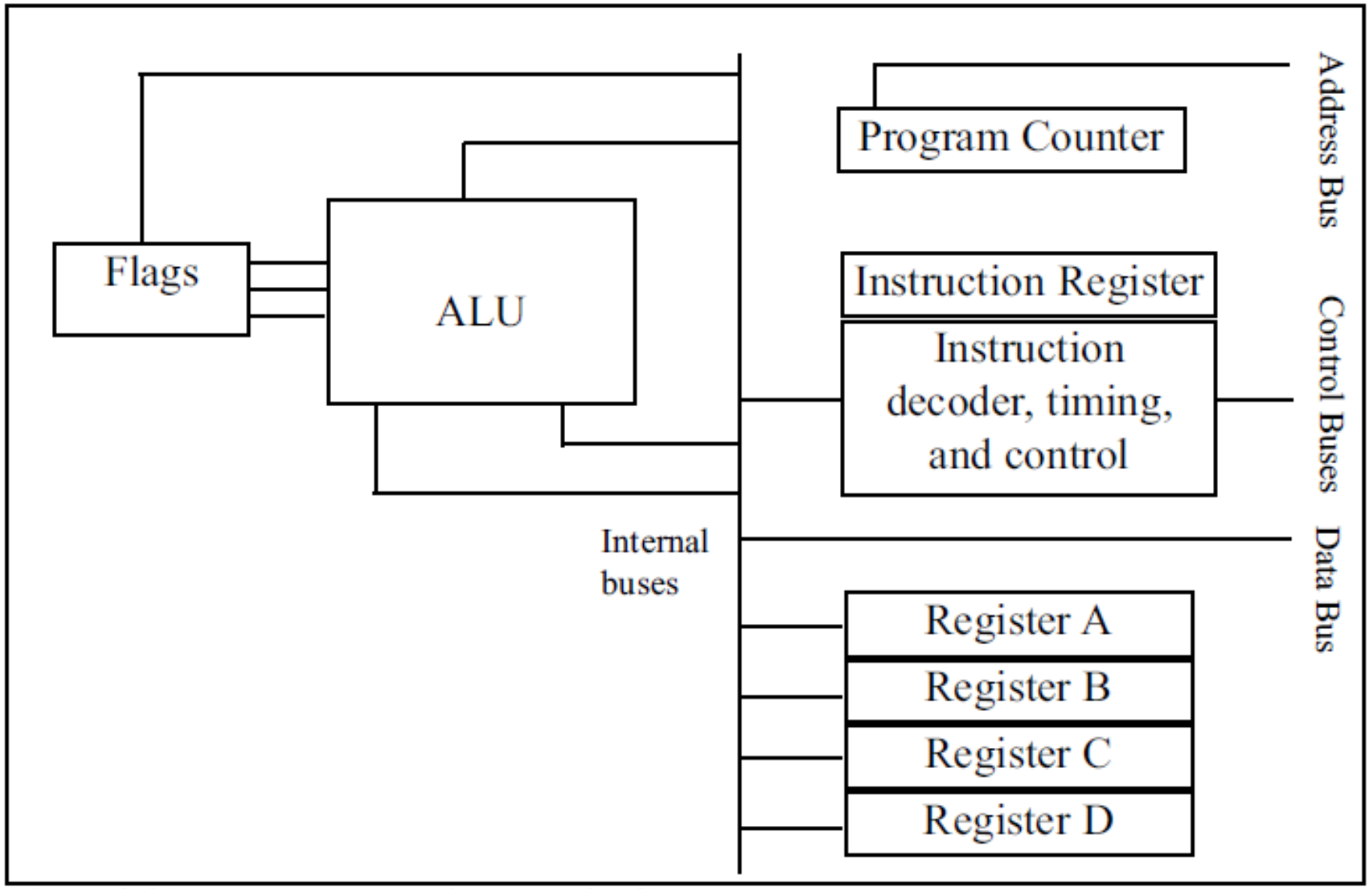
Inside CPU

A program stored in the program memory provides instructions to the CPU to perform actions and tasks.

- An action can simply be adding data such as payroll data or controlling a machine such as a robot.

The function of the CPU is to fetch these instructions from the program memory and execute them. To perform the actions of fetch and execute, all CPUs are equipped with the following components:

Inside CPU


Flags
ALU
Program Counter
Instruction Register
Instruction decoder, timing, and control
Internal buses
Register A
Register B
Register C
Register D
Address Bus
Control Buses
Data Bus

Inside CPU

a) Registers: The CPU uses registers to store information temporarily. The information could be two values to be processed, or the address of the value needed to be fetched from memory. Registers inside the CPU can be 8-bit, 16-bit, 32-bit, or even 64-bit registers, depending on the CPU. In general, the more and bigger the registers, the better the CPU.

b) Arithmetic/logic unit (ALU): The ALU section of the CPU is responsible for performing arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT.

c) Program counter: The function of the program counter is to point to the address of the next instruction to be executed.

- As each instruction is executed, the program counter is incremented to point to the address of the next instruction to be executed.
- The contents of the program counter are placed on the address bus to find and fetch the desired instruction.

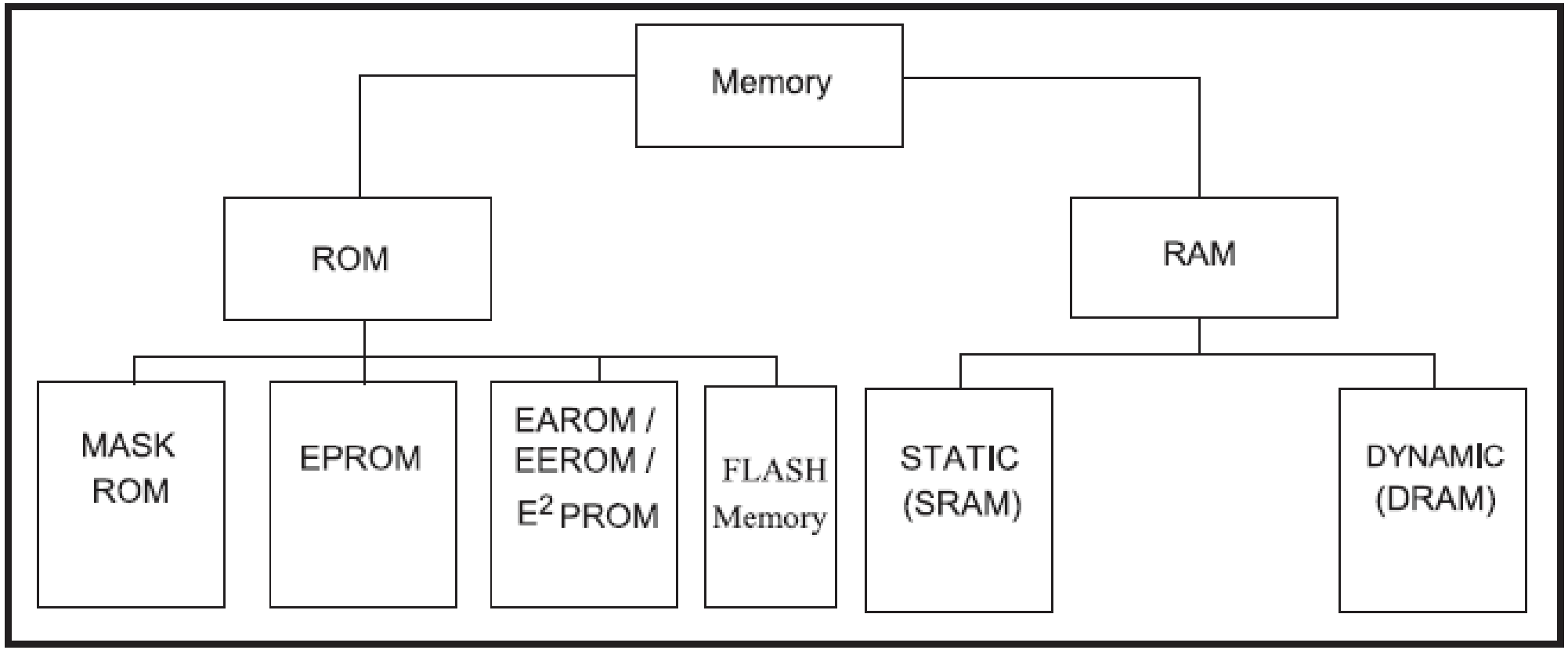
Inside CPU

d) Instruction register: The function of the instruction register is to store (fetch) an instruction loaded from the program memory.

e) Instruction decoder: The function of the instruction decoder is to interpret the instruction fetched into the CPU. One can think of the instruction decoder as a kind of dictionary, storing the meaning of each instruction and what steps the CPU should take upon receiving a given instruction.

f) Flags: They show status of CPU operation such as carry bit from addition.

Types of Memory

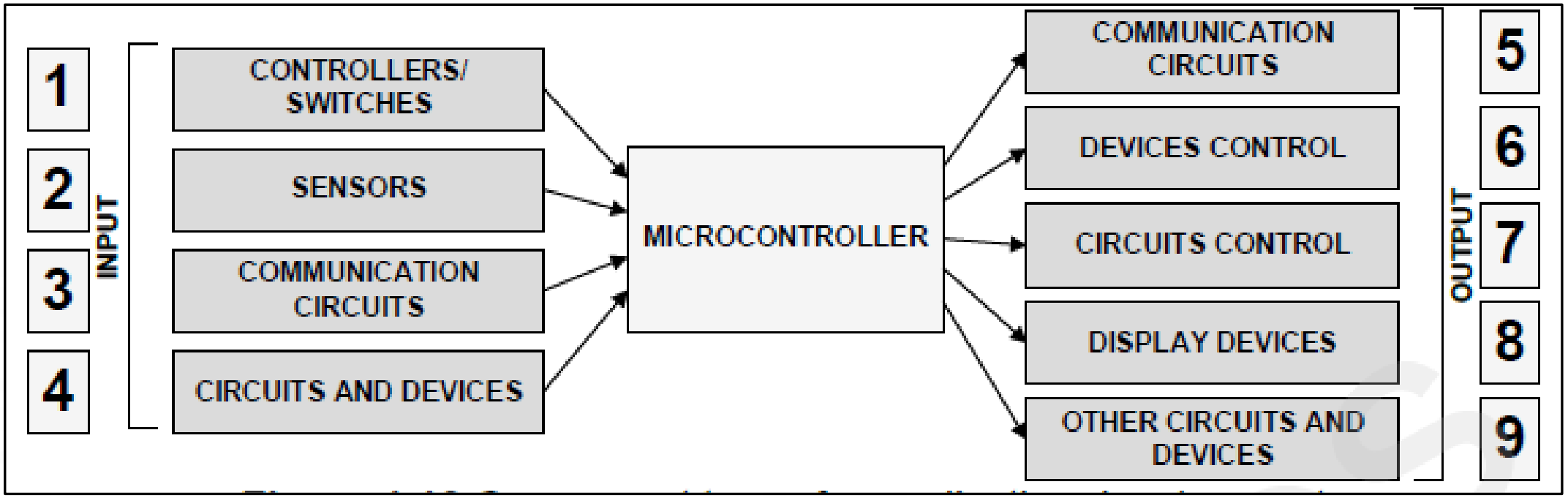


Types of Memory

- ROM (read-only memory) is a type of *memory that does not lose its contents when the power is turned off*. For this reason, ROM is also called nonvolatile memory. There are different types of read-only memory, such as PROM, EPROM, EEPROM, Flash EPROM, and mask ROM.
  - *The program memory is a ROM*.
- RAM (random access memory) memory is called *volatile memory since cutting off the power to the IC results in the loss of data*. Sometimes RAM is also referred to as RAWM (read and write memory), in contrast to ROM, which cannot be written to. There are two popular types of RAM: static RAM (SRAM) and dynamic RAM (DRAM).
  - *The data memory is a RAM*.

Microcontroller Applications

A microcontroller operates autonomously like a “small computer”. The development of a full functional application which is fully controlled by a microcontroller, requires the combination of the most suitable external circuits, sensors, etc. The figure below shows a set of external “components” that can be combined in order to build a desired application.

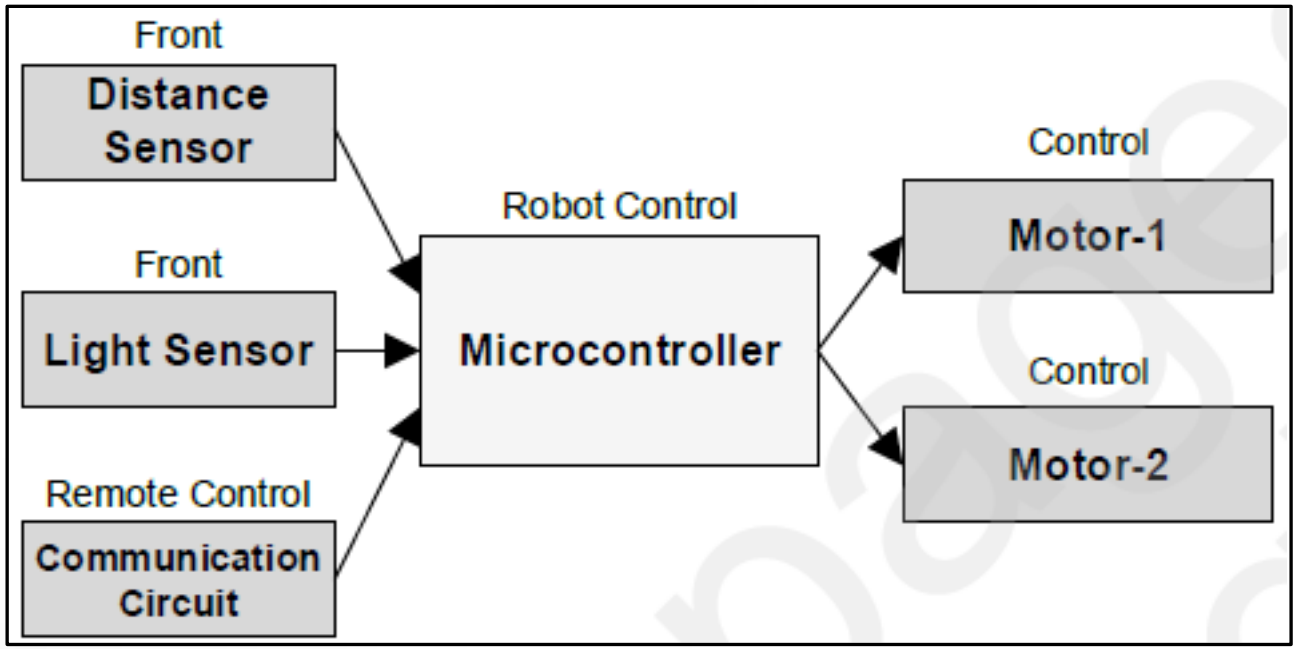


Microcontroller Applications

- Temperature display: This application measures the room temperature via a temperature sensor and show the value on a seven-segment display.

Microcontroller Applications

- Controlling a robot: the "control center" of the robot controls the motor's activation according to signals that are sent by the light and distance sensors. The microcontroller represents the "control center" and is programmed to make a combined processing of the sensors measurements and the remote control instructions in order to decide for the robot movement. The movement is controlled by small signals that are sent from the microcontroller to the motor control circuits.



2. Programming Languages

- To control a microcontroller, we must write a program and upload it into the microcontroller (i.e., programmable).
- Originally, a microcontroller must be programmed by the assembly (low-level) language.
- Many microcontrollers support high-level programming languages such as C, Java, Python.

Machine Language

A microcontroller reads and executes a set of *machine language instructions* (program) represented by binary numbers (0, 1) to perform a desired job.

- A machine language instruction might be shown as a binary number or a hex number for human understandable.

0000111000000010		0E02
0000111100000011	or	0F03
0110111001000000		6E40
1110111100000011		EF03

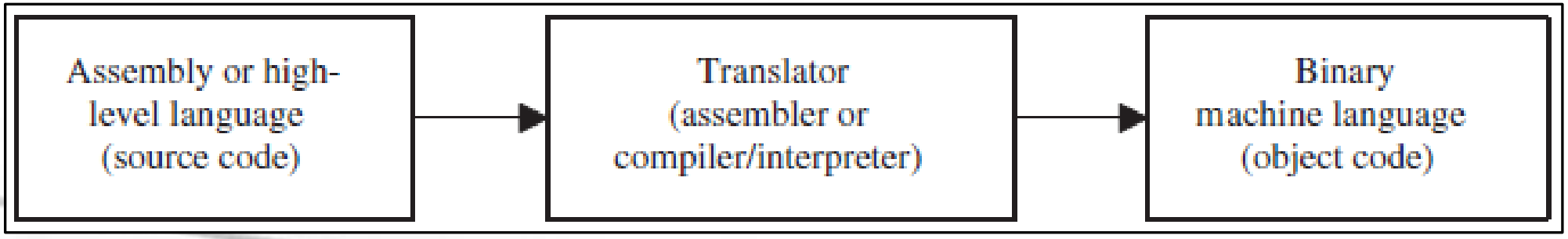
- The program is stored in the program memory.
- A microcontroller has a unique set of machine language instructions defined by its manufacturer. No two microcontrollers by different manufacturers have the same machine language instruction set.

Assembly Language

Microcontrollers can be programmed using semi-English-language statements called the *assembly language*. However, a translator must be used to convert such programs into binary machine language so that the microcontroller can execute the programs.

```
lds     keyCode, lastdigit
sbr     temp, 1<<7              ;set bit 7 for event
or      keyCode, temp
ldi     temp, $0F               ;clear last digit
sts     lastdigit, temp
```

- Assembly and high-level language programs are called *source codes*.
- Machine language programs are known as *object codes*.
- A translator converts source codes to object codes.



Why Study Assembly Language?

Assembly language programming is important in the understanding the internal architecture of a microcontroller and may be useful for debugging programs.

- The study of assembly language should not be undertaken with the goal of simply learning another programming language.
- Instead, this should be seen as laying a solid foundation to support the programming knowledge and skills already acquired.
- Studying assembly language will make you more aware of how computer systems work and how high-level language programming is used to control them.
- Knowledge about the details of a specific processor gained through the study of assembly language programming will ultimately make you a better computer practitioner.

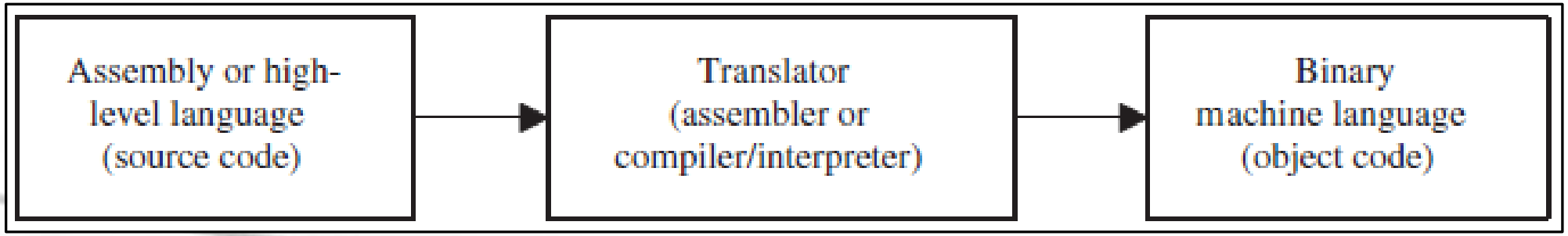
High-Level Languages

High-level language programs composed of English-language-type statements rectify all these deficiencies of machine and assembly language programming. *The programmer does not need to be familiar with the internal microcontroller structure or its instruction set*. Also, each statement in a high-level language corresponds to a number of assembly or machine language instructions. As a result, *to perform a same job, a high-level language program is shorter*.

For the Arduino board, two high-level languages are available:

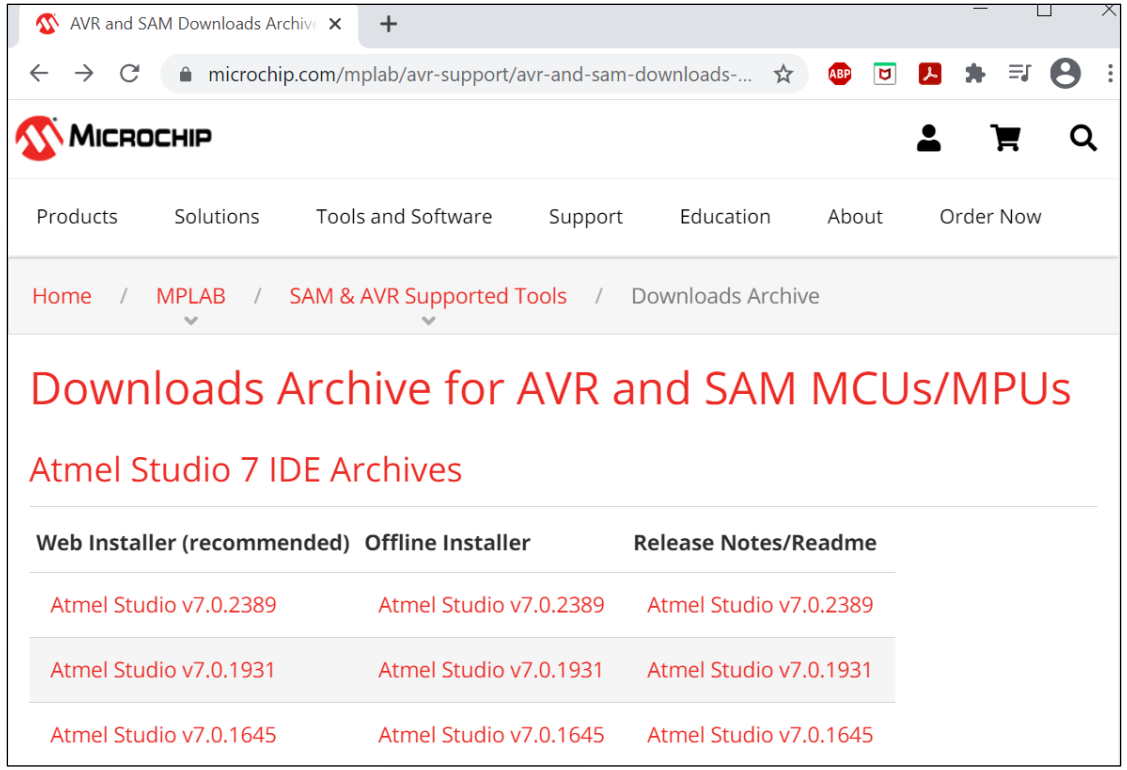
- C programing language
- Arduino sketch programing language (very popular and easy).

Similarly, the compiler/interpreter will convert the source code to the object code.



Development Software

- We will use the Atmel Studio 7 as our development software to write, build and debug our applications written in C/C++ or assembly code.
- The Atmel Studio 7 can be downloaded from this link.
  https://www.microchip.com/mplab/avr-support/avr-and-sam-downloads-archive



Example, ATmega328 Microcontrollers

A List of Microcontrollers in the Market

Microcontrollers can be classified according to the size of the data bus of the data memory, which identify how many bits of data can be loaded at once. We have 8-bit, 16-bit, and 32-bit microcontrollers as shown below. Note that the *ATmega328 is an 8-bit microcontroller*.

Table 1-2A: Some Microcontrollers

8-bit

- 8051
- AVR (Microchip, formerly Atmel)
- PIC16, PIC18 (Microchip)
- S08 or 64HCS08 (NXP, formerly Freescale)

16-bit

- PIC24, dsPIC (Microchip)
- HCS12 (NXP/Freescale)
- MSP430 (TI)

32-bit

- ARM
- AVR32 (Atmel)
- ColdFire (Freescale)
- MIPS32
- PIC32 (Microchip)
- PowerPC
- SuperH
- TriCore (Infineon)

Atmel AVR® Microcontrollers

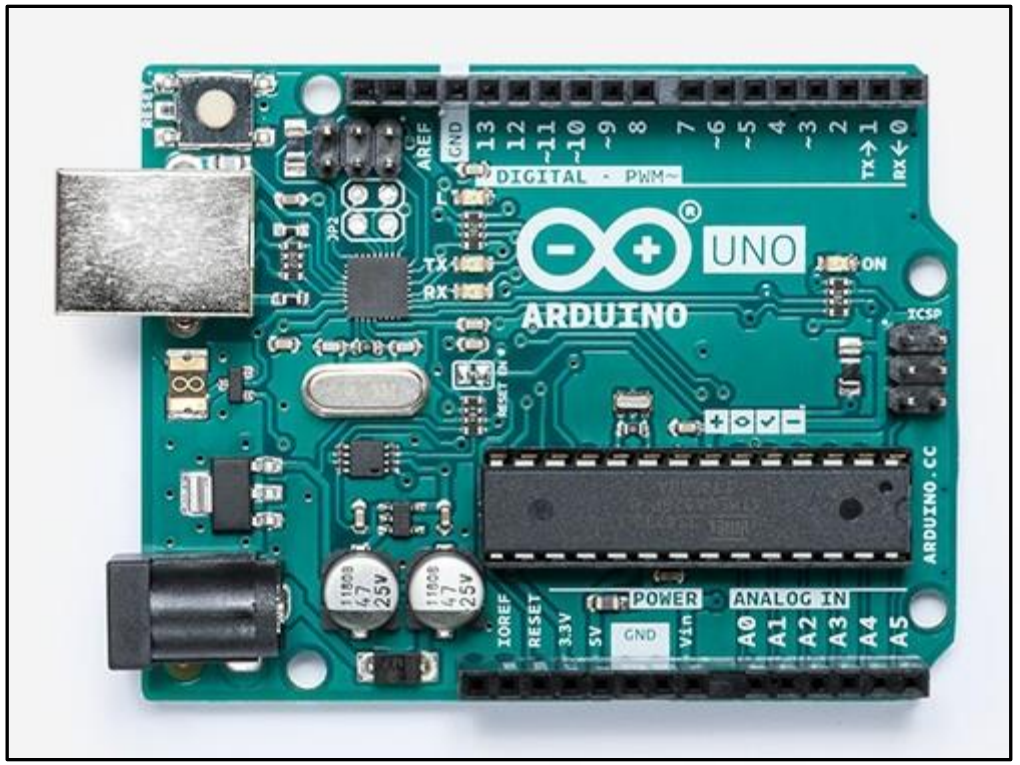
ATmega328 is an Atmel AVR® microcontroller developed by a company named Microchip Technology. There are many versions of Atmel AVR® microcontrollers depending on the program memory, data memory, EEPROM, etc.

Table 1-4: Some Members of the ATmega Family

Part Num.	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32, PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44, PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44, PDIP40
ATmega328	32K	2K	1K	23	8	3	TQFP32,PDIP28
ATmega64	64K	4K	2K	54	8	4	TQFP64, MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100, CBGA
ATmega2560	256K	4K	8K	86	16	6	TQFP100, CBGA

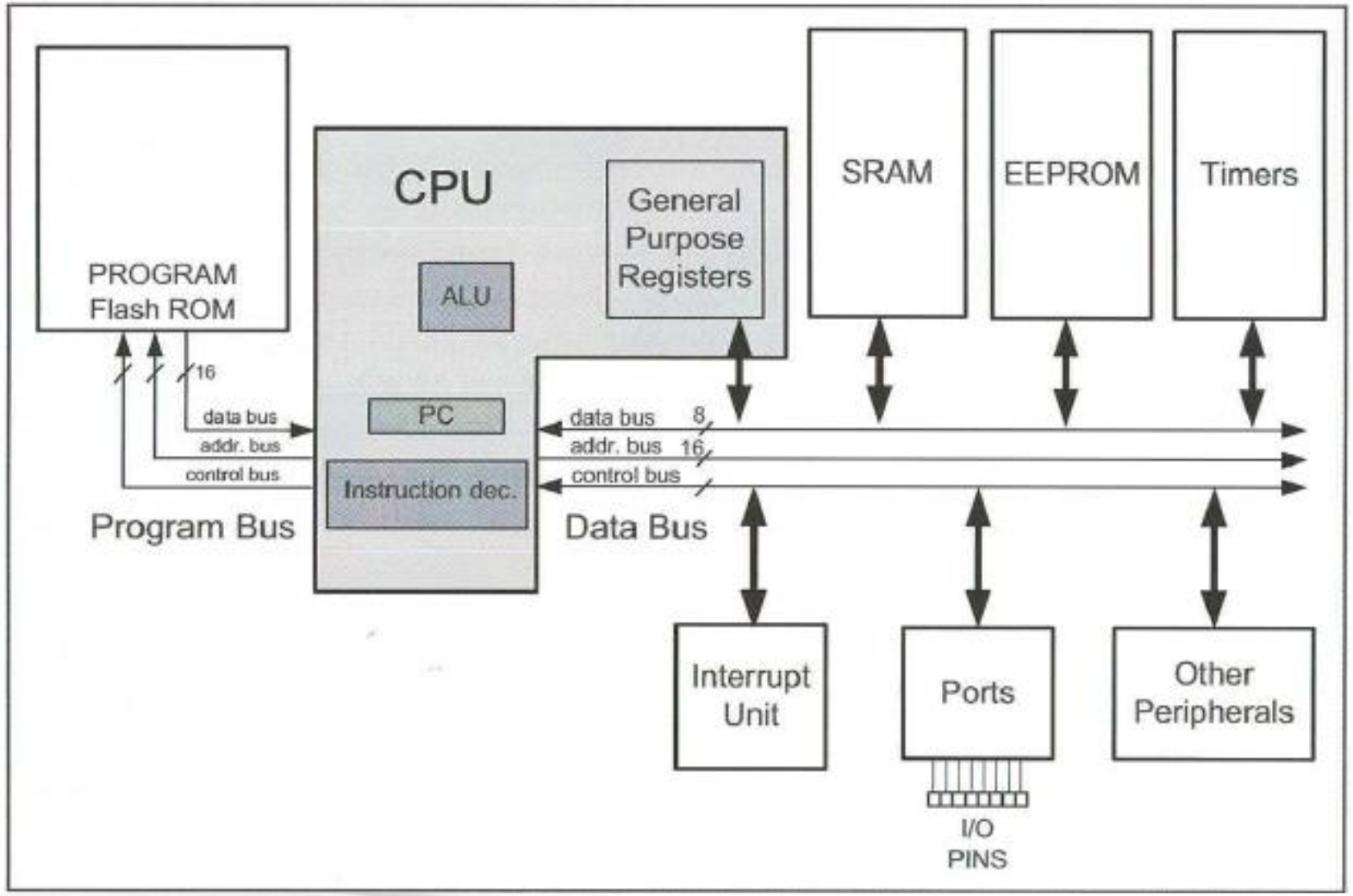
Our Development Board: Arduino UNO

ATmega328 is also used as the microcontroller in the Arduino UNO board (which will be used our development board).



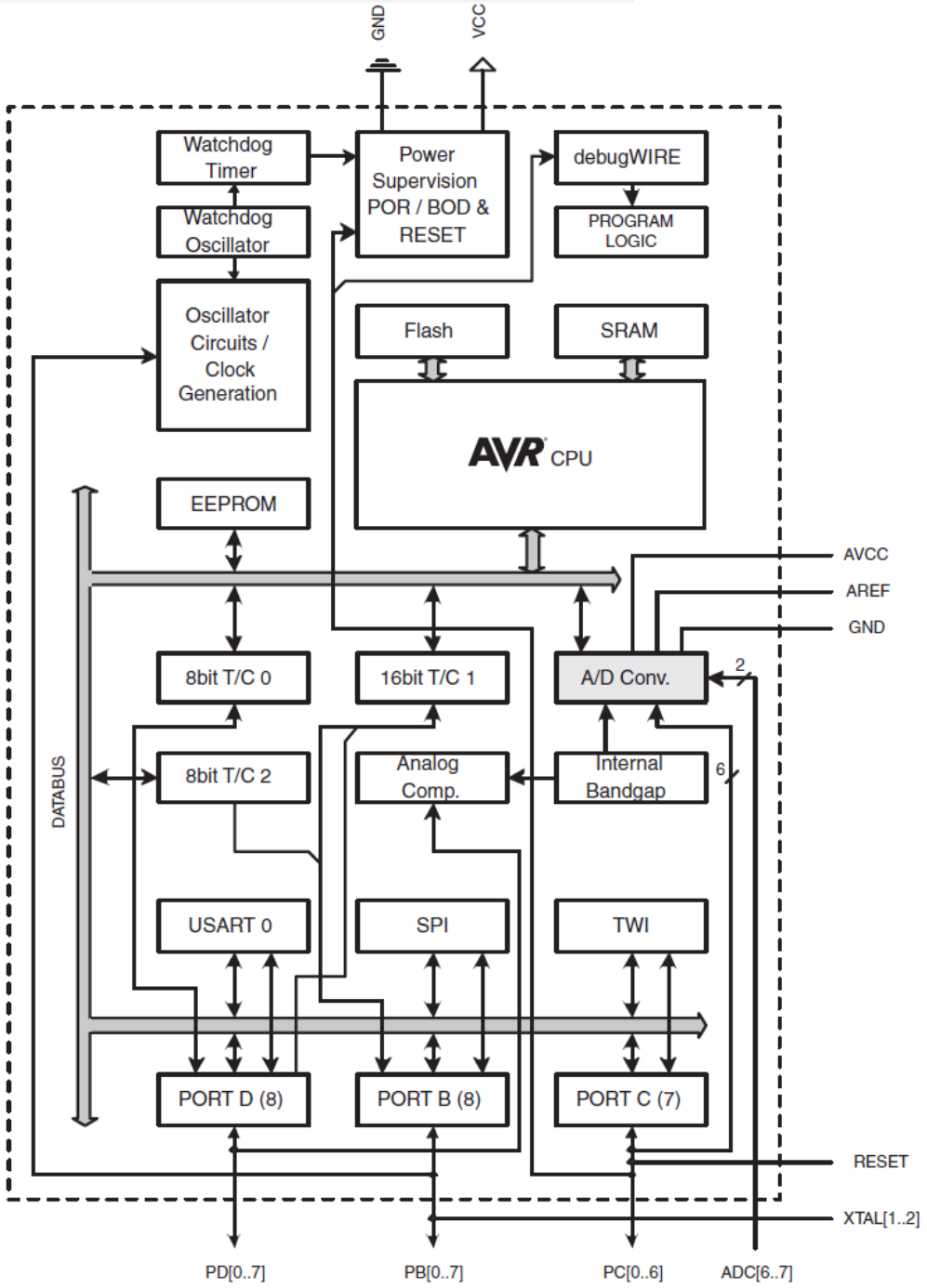
Basic AVR® Architecture

The AVR® microcontroller architecture is shown below. It uses *Harvard architecture*, which means there are separate buses for the program memory and the data memory.



ATmega328 Microcontroller – Block Diagram

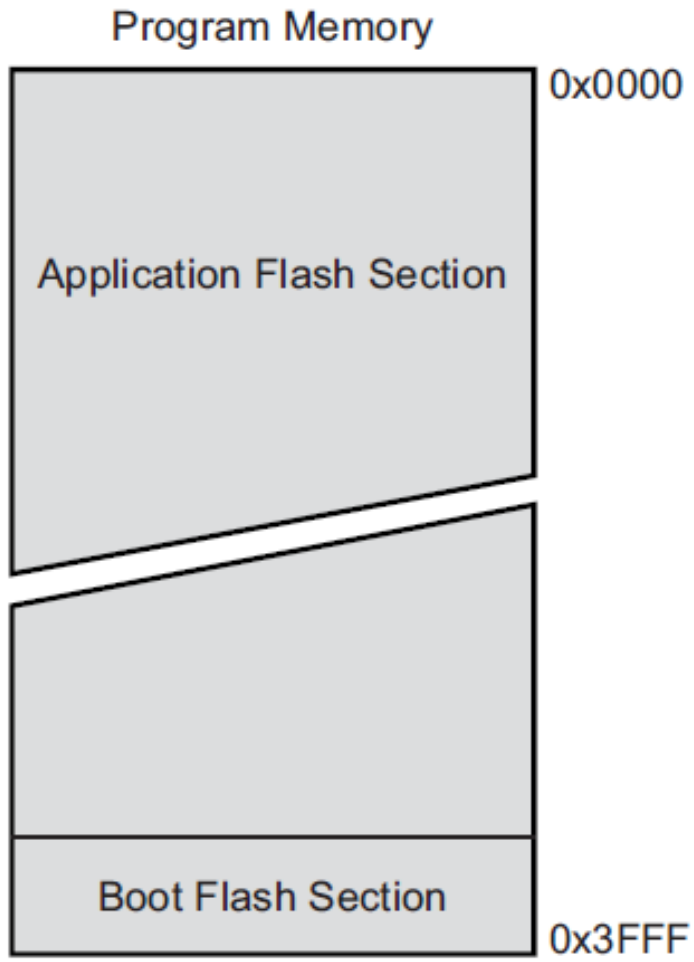
This block diagram shows the main components of ATmega328 microcontrollers. We will study many of them later in the course.



ATmega328 – Program Memory

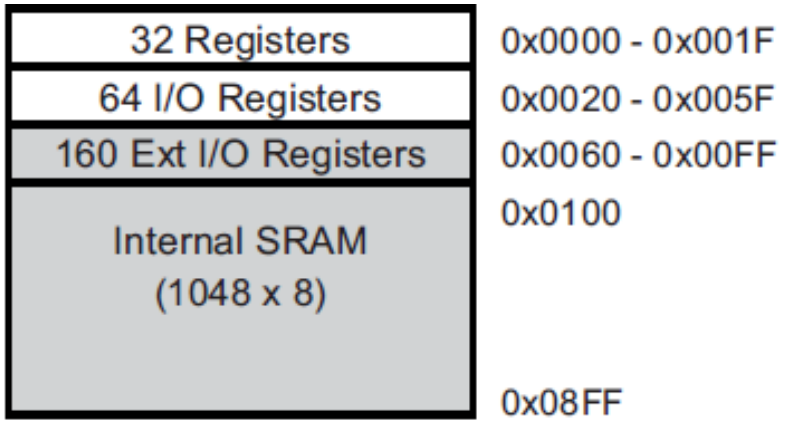
The ATmega328 program memory has the size of 32Kbytes, which is a reprogrammable flash ROM.

- It stores the program (in the form of machine instructions – binary numbers).
- This program memory is organized as 16K × 16bits, which means
  - The program memory is divided into 16 × 1024 slots (= 16,384 slots).
  - Each slot has an address identified by a hex number starting form 0x0000. As a result, we have addresses from 0x0000 to 0x3FFF (=16,384 addresses).
  - Each slot stores 16 bits (=2 bytes).
- The data bus between program memory and CPU will carry 16 bits per time.



ATmega328 – Data Memory

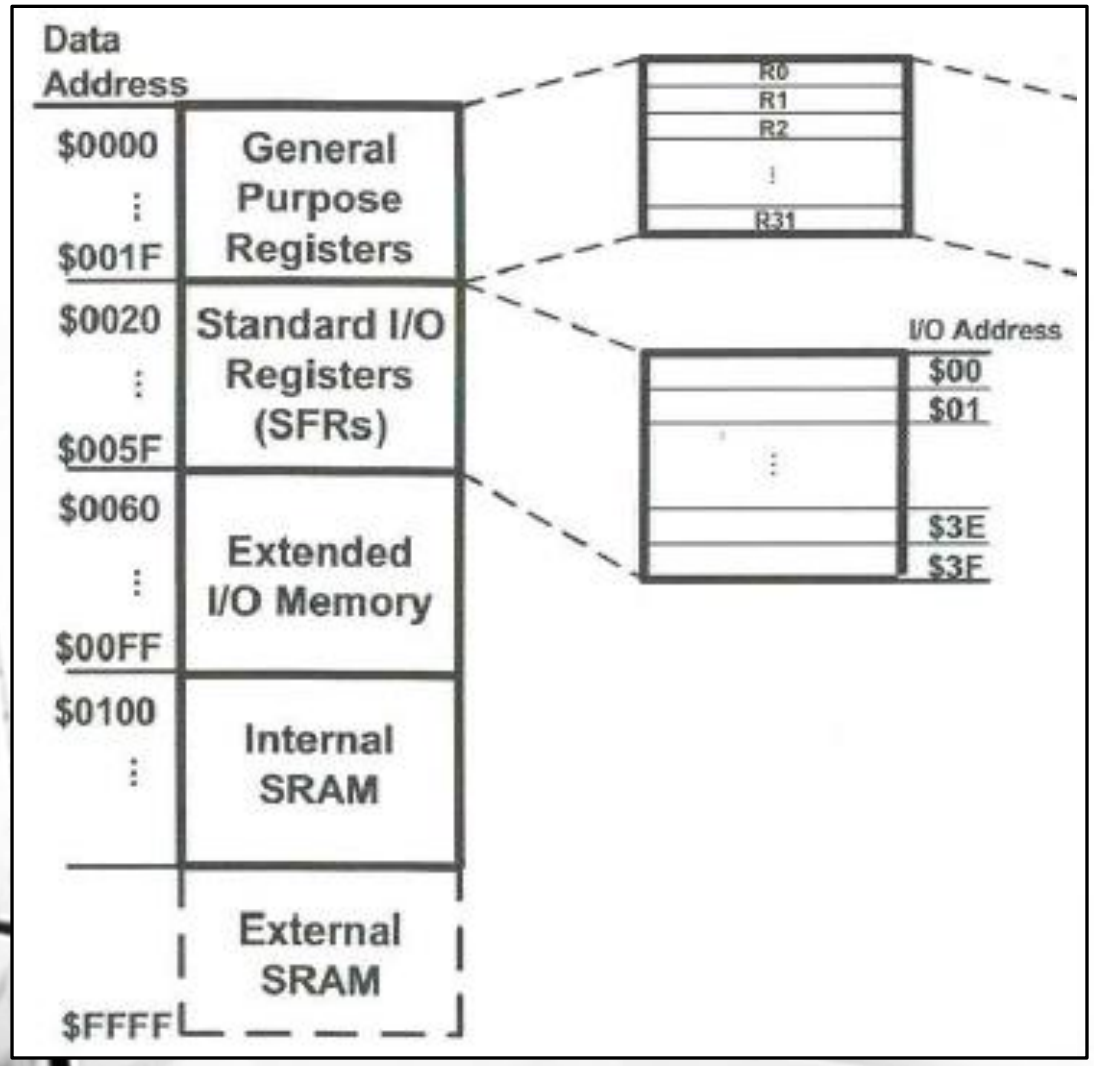
- The *data memory* of ATmega328 consists of 4 sections. Note that each slot (=each address) stores 8 bits.
  - 32 general propose registers (GPRs) – addresses 0x0000 – 0x001F. (8 bits per address)
  - 64 I/O registers – addresses 0x0020 – 0x005F. (8 bits per address)
  - 160 extended I/O registers – addresses 0x0060 – 0x00FF. (8 bits per address)
  - Internal SRAM with size 2Kbytes. Then, we have 1048 slots (= addresses) – addresses 0x0100 – 0x08FF.
- The data bus between data memory and CPU will carry 8 bits per time.
- General propose registers (GPRs) are inside the CPU.
- SRAM is used to store temporary data obtained from running a program.
- What are stored in both memory will be binary numbers (0s and 1s). Sometime, we show/represent these data in a form of a *hex number* for convenience.



Example, ATmega328 Data Memory

Data Memory Structure and Addresses

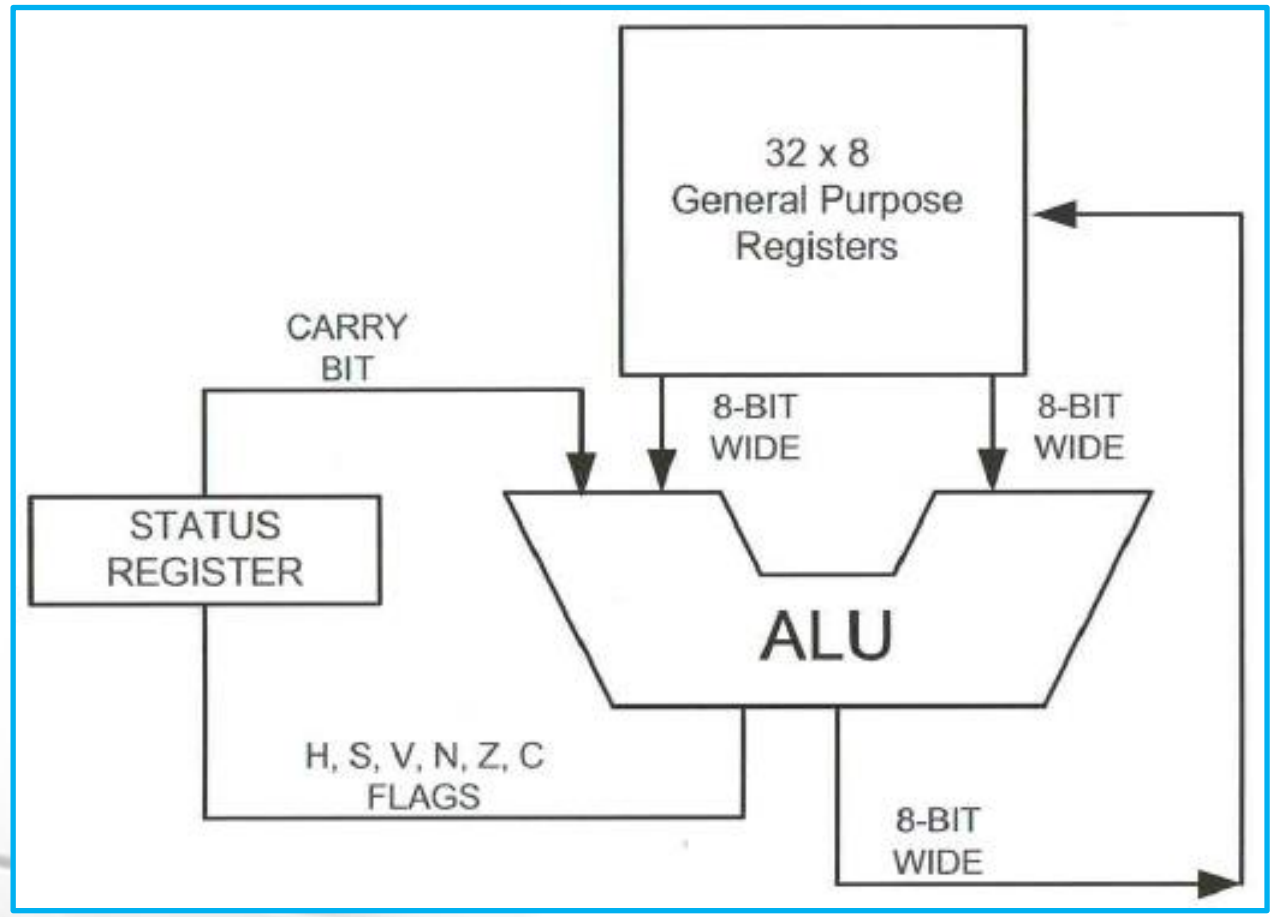
The *data memory structure and addresses* of the AVR microcontroller is shown below. Each address contains 8 bits or 1 byte.



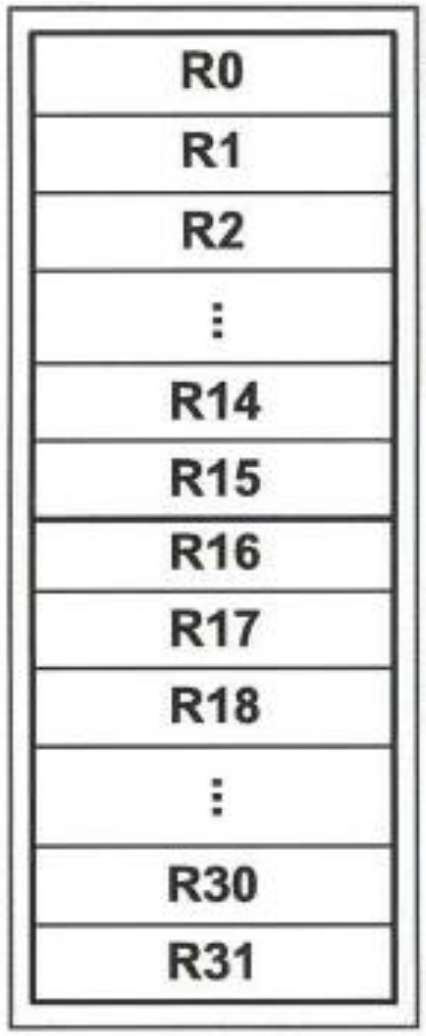
- It consists of general propose registers (GPRs), I/O registers, extended I/O memory, SRAM, and external SRAM (if available).
- The data memory is divided into (unique) data addresses.
- Each data address stores an 8-bit binary number (= 1 byte).

General Propose Registers

General propose registers (GPR) are data memory inside the CPU and connected to the arithmetic/logic unit (ALU). They are used to store the data which will be inputted/outputted to ALU. Note that the ALU is used to do arithmetic (add, subtract, etc.) and logic (OR, AND, etc.)

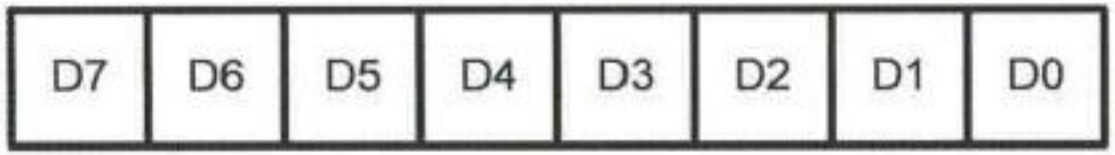


General Propose Registers



The GPR structure is shown below.

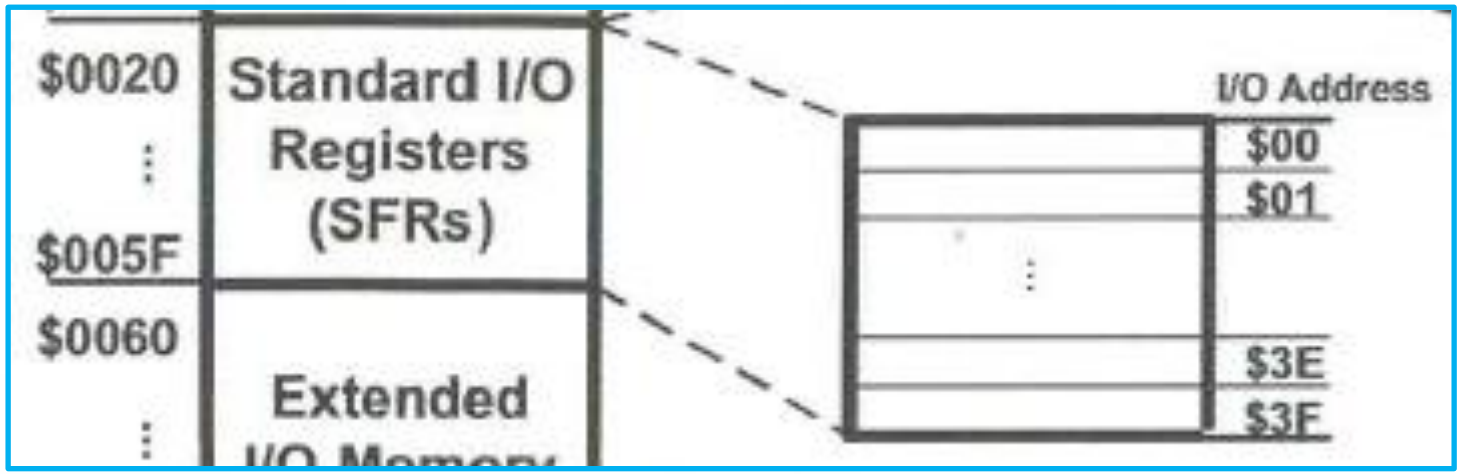
- There are 32 registers.
- Each register is identified by an address ($0000 - $001F) or by a name (R0 – R31).
- Each register stores 8 bits: MSB → D7 and LSB → D0.



I/O Registers

The AVR microcontroller has 64 I/O registers. Note that each register contains 8 bits or 1 byte. So, the capacity of I/O memory is 64 bytes.

- An address of an I/O register can be identified by the memory address or the I/O address.
  - The *memory addresses* are specified as $0020 - $005F (there are 64 addresses).
  - The *I/O addresses* are specified by $00 - $3F (there are 64 addresses) or their names.



I/O Registers

The I/O memory is dedicated to store the data of I/O ports or pins (to read/write), status register, timers, ADC, serial communication, etc.

- There are 64 registers.
- This figure shows the memory addresses, I/O addresses and I/O names.
- More details are studied in Chapter 4.

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

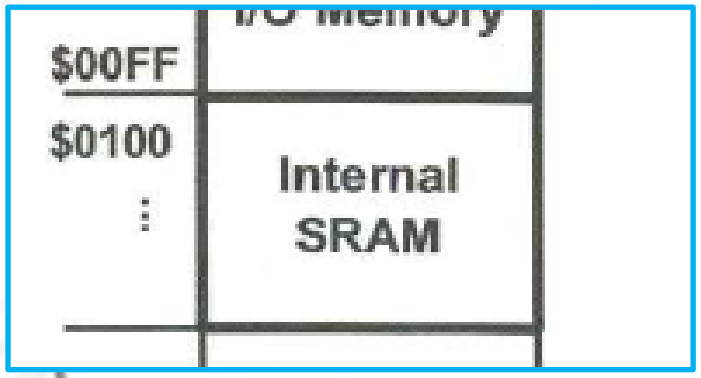
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

SRAM

SRAM stores temporary data/values obtained from program executions for example, stack values.

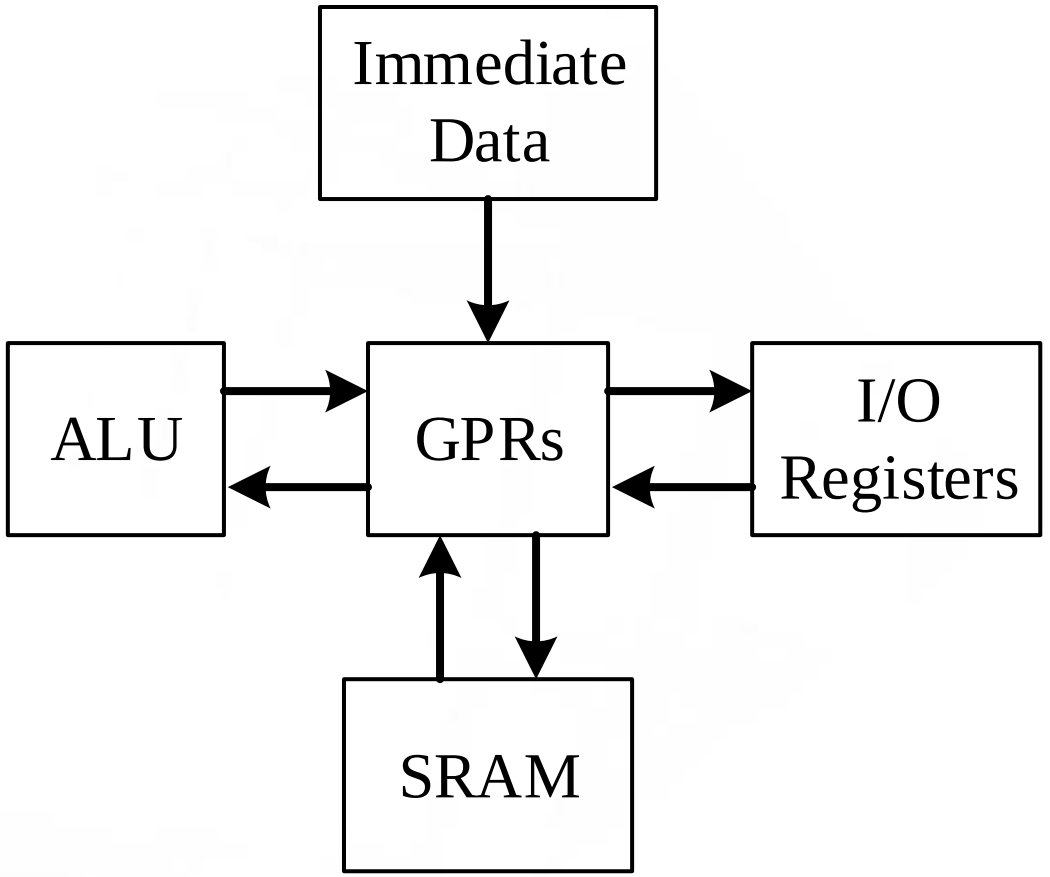
- The data address in SRAM starts from the address $0100.
- Since the ATMEGA328 has a 2048-byte SRAM, the last address in SRAM will be $08FF.
  - 2048 bytes → 2048 addresses (one address contains 8 bits or 1 byte).
  - 2048 → $0800 addresses
  - $00FF+$0800 = $08FF



	Data Memory (Bytes)	=	I/O Registers (Bytes)	+	SRAM (Bytes)	+	General Purpose Register
ATtiny25	224		64		128		32
ATtiny85	608		64		512		32
ATmega8	1120		64		1024		32
ATmega16	1120		64		1024		32
ATmega32	2144		64		2048		32
ATmega328	2304		64+160		2048		32
ATmega128	4352		64+160		4096		32
ATmega2560	8704		64+416		8192		32

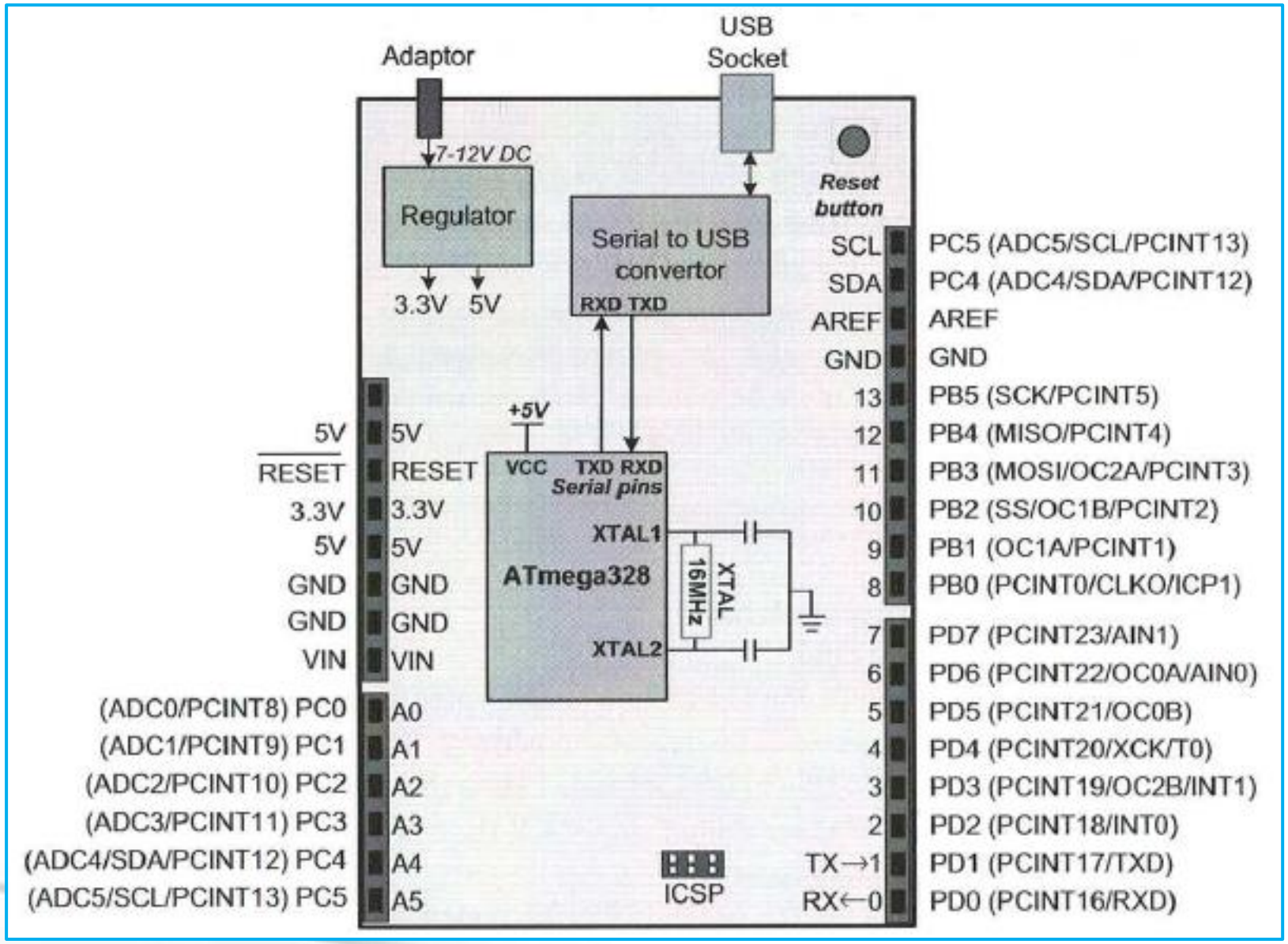
Relationship among ALU, GPRs, I/O Registers, and SRAM

The way that a data value will be stored can be displayed as the figure below. The GPRs will be the middleman who relays the data among ALU, I/O registers, and SRAM.

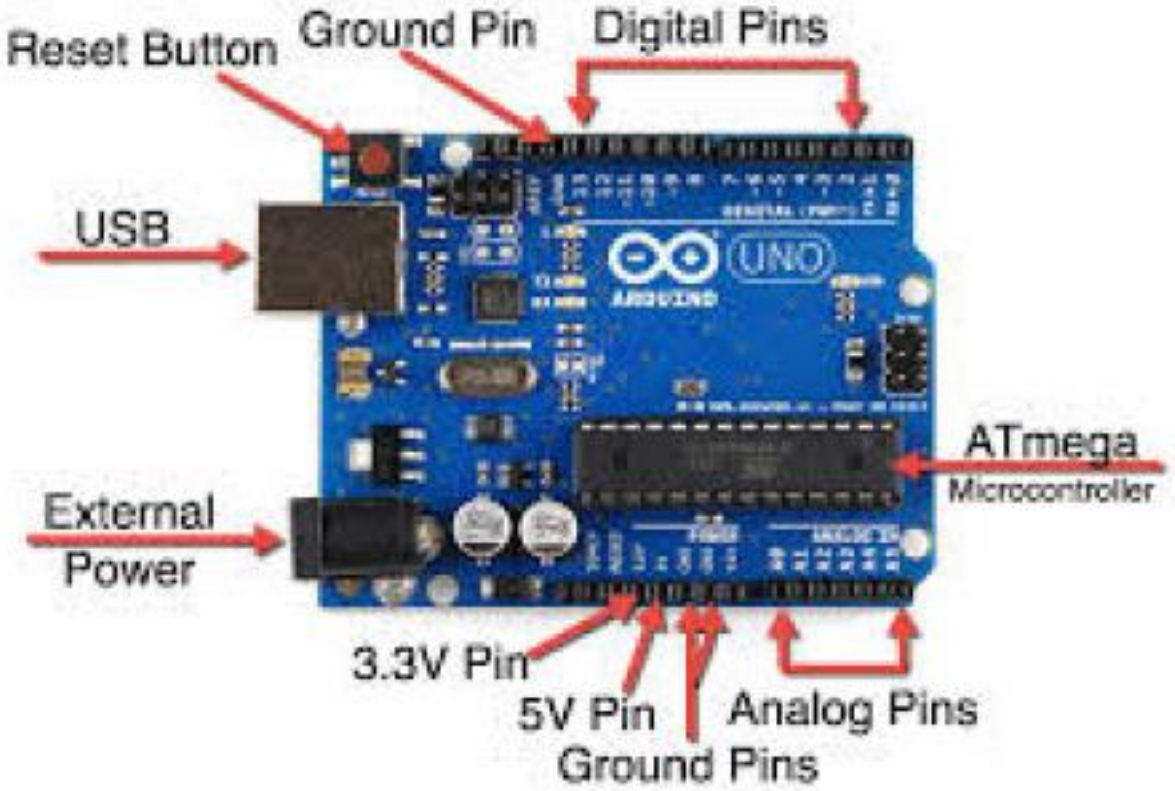


Introduction to Arduino Programming using Arduino IDE

Pin Layout of Arduino UNO



Arduino UNO Layout



Arduino UNO interfaces:

- 14 digital input/output (I/O) pins: 1 – 13
- 6 analog inputs pins: A0 – A5
- 6 PWM pins: pins with the sign ~
- Communications: USART (Tx and Rx pins), I2C, SPI

List of Arduino UNO Pins

Label	Description
IOREF	Provides the reference voltage used by the microcontroller if not 5V.
RESET	Resets the Arduino when set to LOW.
3.3V	Provides a reduced 3.3V for powering low-voltage external circuits.
5V	Provides the standard 5V for powering external circuits.
GND	Provides the ground connection for external circuits.
GND	A second ground connection for external circuits.
Vin	An external circuit can supply 5V to this pin to power the Arduino, instead of using the USB or power jacks.
A0	The first analog input interface.
A1	The second analog input interface.
A2	The third analog input interface.
A3	The fourth analog input interface.
A4	The fifth analog input interface, also used as the SDA pin for TWI communications.
A5	The sixth analog input interface, also used as the SCL pin for TWI communications.

List of Arduino UNO Pins

Label	Description
AREF	Alternative reference voltage used by the analog inputs (by default, 5V).
GND	The Arduino ground signal.
13	Digital port 13, and the SCK pin for SPI communication.
12	Digital port 12, and the MISO pin for SPI communication.
~11	Digital port 11, a PWM output port, and the MOSI pin for SPI communications.
~10	Digital port 10, a PWM output port, and the SS pin for SPI communication.
~9	Digital port 9, and a PWM output port.
8	Digital port 8.
7	Digital port 7.
~6	Digital port 6, and a PWM output port.
~5	Digital port 5, and a PWM output port.
4	Digital port 4.
~3	Digital port 3, and a PWM output port.
2	Digital port 2.
TX -> 1	Digital port 1, and the serial interface transmit port.
RX <- 0	Digital port 0, and the serial interface receive port.

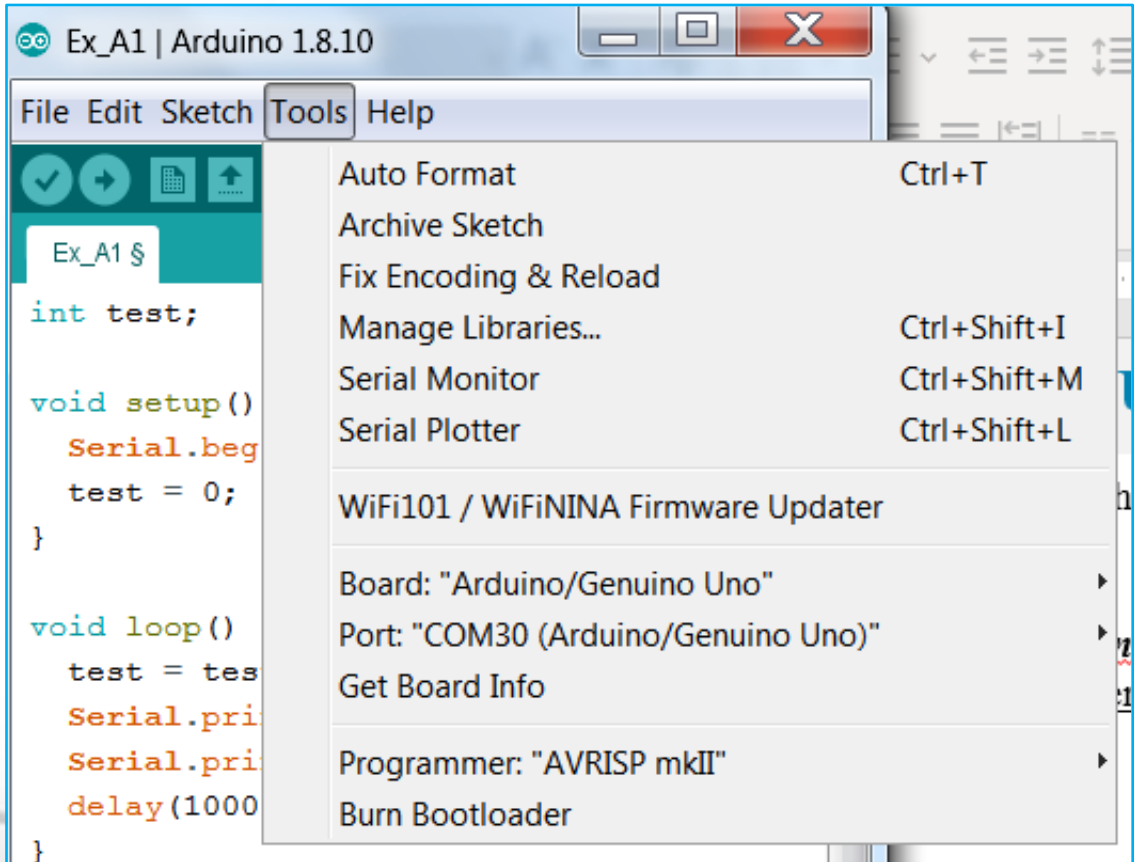
Arduino IDE

- We will write a program to control the Arduino UNO by using the software called the Arduino IDE.
- The programing language used here is called the Arduino programming language.
  - It is a variant of C++ (also similar to C).
- A program that we wrote to control the Arduino UNO is also called a sketch.
- The software Arduino IDE can be download from this link:

https://www.arduino.cc/en/main/software

Setting up Arduino IDE: Arduino UNO

1. Connect Arduino UNO to the computer. Record the COM port number (see from the device manager).
2. Open the Arduino IDE software.
3. Click ***Tools*** > ***Board*** and choose ***Arduino/Genuino UNO***
4. Click ***Tools*** > ***Port*** and choose COM port number from Step 1.



Write an Arduino Sketch

File name

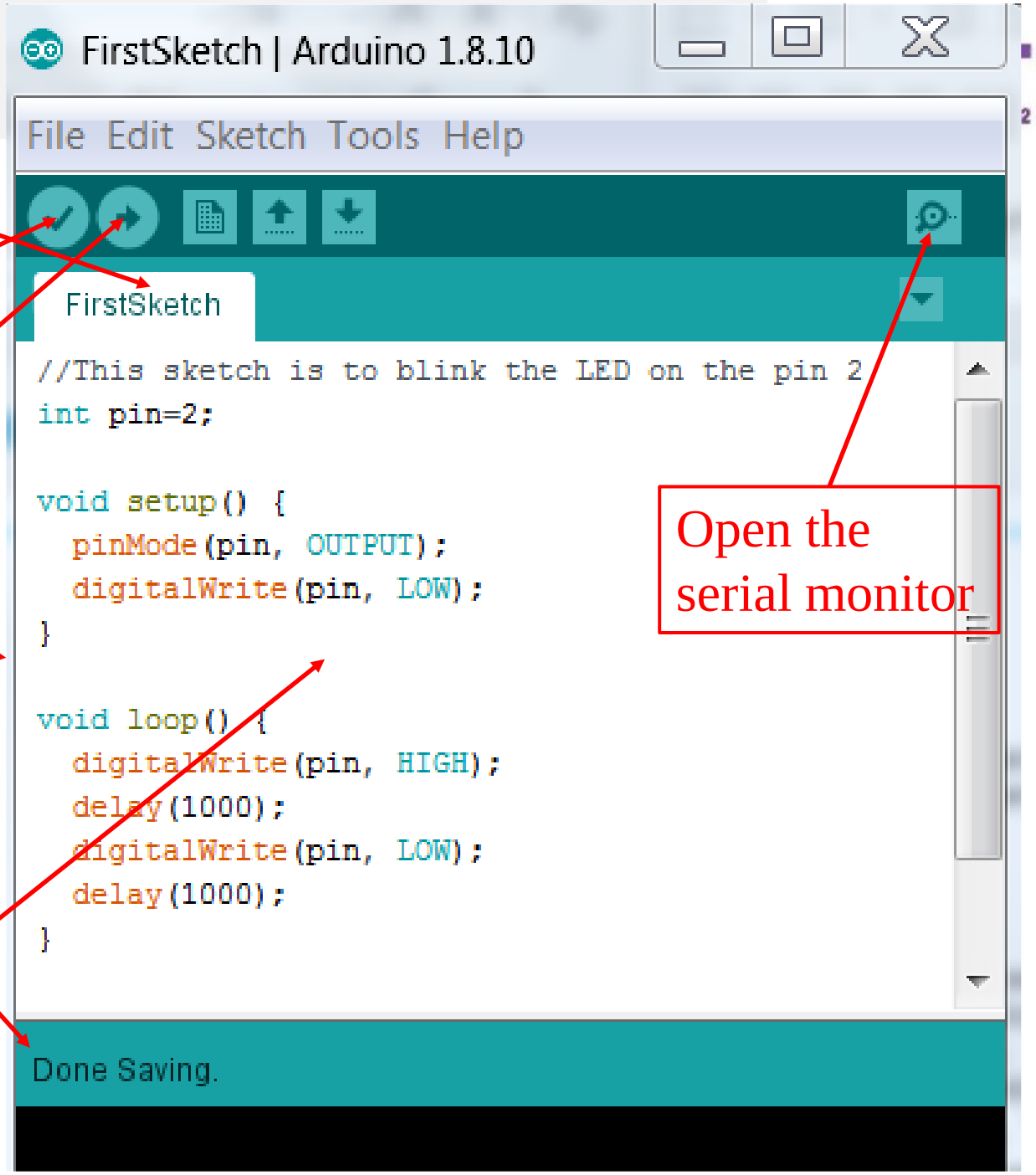
To verify the sketch

To upload the sketch to Arduino UNO

Write a sketch in this area.

Status of after verifying/uploading the sketch

This sketch will turn the LED on for 1 second, and off for 1 second.



Open the serial monitor

Uploading a Sketch

1. Write an Arduino sketch on the Arduino IDE.
2. If we want to verify the sketch, we click the Verify button.
3. If we want to upload the sketch to the Arduino UNO, we click the Upload button. This step will verify the sketch and if there are no errors, the sketch will be uploaded to the Arduino UNO.

2. Arduino Sketch Layout/Structure

Arduino Sketch Layout

An Arduino sketch (program/code) basically consists of the following parts. We read the code here from top to bottom, line-by-line.

1. Header files:
   - Arduino provides external library files for specific hardware/commands, which we can use functions in these libraries immediately by using the ***#include*** directive.
   - We put them at the beginning of the sketch.
2. Global variable declaration:
   - We declare global variables and their data types here.

```
#include <header file>

Global variable declaration

void  setup() {
      command;
      ...
      command;
}

void  loop() {
      command;
      ...
      command;
}
```

Arduino Sketch Layout

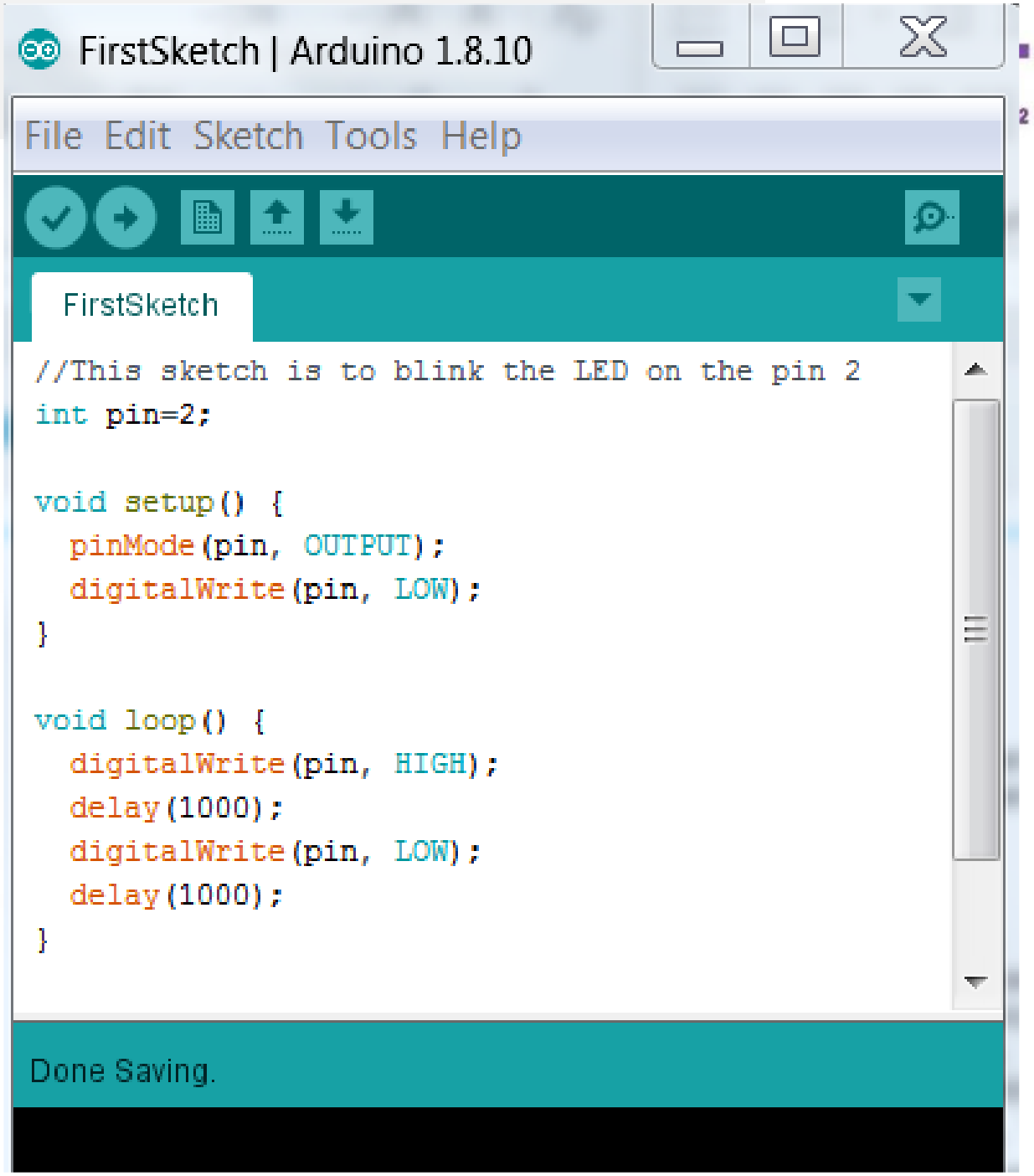
3. The ***setup()*** function:
   - When power on, the Arduino will start the bootloader (basically, Arduino Operating System). The bootloader specifically looks for two functions: ***setup()*** and ***loop()***.
   - The commands in the ***setup()*** function are those we want to set up the Arduino board and will run only one time at the start.
4. The ***loop()*** function:
   - Our main code will be in this function.
   - The commands in this function will be repeated until the Arduino UNO is turned off.

```
#include <header file>

Global variable declaration

void  setup() {
      command;
      ...
      command;
}

void  loop() {
      command;
      ...
      command;
}
```

Example

This sketch controls the Arduino UNO to on (1) and off (0) the pin 2, every 1 second.

1. Declare a global variable called pin and, initially, set to 2.
2. In the ***setup()*** function, we set the pin mode and let the pin 2 be low (0).
3. In the ***loop()*** function, we alternatively, set the pin 2 to be high (1) and low (0), every 1 second.



3. Basic Arduino Programming Commands

Basic Arduino Programming Commands

1) Working with variables
2) Math commands
3) Serial monitor
4) Creating a time delay
5) Working with comparison conditions
6) Using the ***if***, ***if-elseif***, and ***else if*** commands
7) Using the ***while*** and ***do-while*** loops
8) Using the ***for*** loop
9) Creating a function

1) Working with Variables

Before being used, a variable must be declared in the following format.

```
datatype  Variable_name;
```

Example: To declare a variable name ***pin*** to hold an integer data type (***int***), we write:

```
int   pin;
```

1. Variable names:
   - Contain only letters, numbers, underscores, or dollar sign.
   - Start with a letter.
   - Be case sensitive.
   - No limit on the length.

1) Working with Variables

2. Data types: The data type defines the range of the value stored in a variable and the size of the memory usage. A list of data types is shown below.

Data Type	Size (Bytes)	Value Range
boolean	1	Logic true or false
char	1	–128 to +127
byte	1	0 to 255
int	2	–32,768 to 32,767
word	2	0 to 65,535
long	4	–2,147,483,648 to 2,147,483,647
float	4	–3.4028235E+38 to 3.4028235E+38
double	4	–3.4028235E+38 to 3.4028235E+38
unsigned char	1	0 to 255
unsigned int	2	0 to 65,535
unsigned long	4	0 to 4,294,967,295

1) Working with Variables

3. Variable scope:
   - *Global variables*: We declare a global variable outside any functions such as the ***setup()*** function and the ***loop()*** function.
   - *Local variables*: We declare a local variable inside a function where this variable will be known only within this function.

2) Math Commands

1. Standard math operators:
2. Compound operators: A short way to write a simple math assignment.
   Ex.:

   Count +=1;

   means

   Count = Count + 1;

3. Order of operations: Similar to the standard rules.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement
!	Logical NOT
&&	Logical AND
||	Logical OR
&	Bitwise AND
|	Bitwise OR
<<	Left shift
>>	Right shift

2) Math Commands

4. Advanced math commands:

Function	Description
abs(x)	Returns the absolute value of x
constrain(x, a, b)	Returns x if x is between a and b (otherwise returns a if x is lower than a, or b if x is higher than b)
cos(x)	Returns the cosine of x (specified in radians)
map(x, fromLow, fromHigh, toLow, toHigh)	Remaps the value x from the range fromLow to fromHigh to the range toLow to toHigh
max(x, y)	Returns the larger value of x or y
min(x, y)	Returns the smaller value of x or y
pow(x, y)	Returns the value of x raised to the power of y
sin(x)	Returns the sine of x (specified in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of x (specified in radians)

3) Serial Monitor

The serial monitor is a terminal window of the Arduino IDE.

- We can program an Arduino UNO to print a message on the serial monitor.
- We also can program an Arduino UNO to receive a message from the serial monitor.

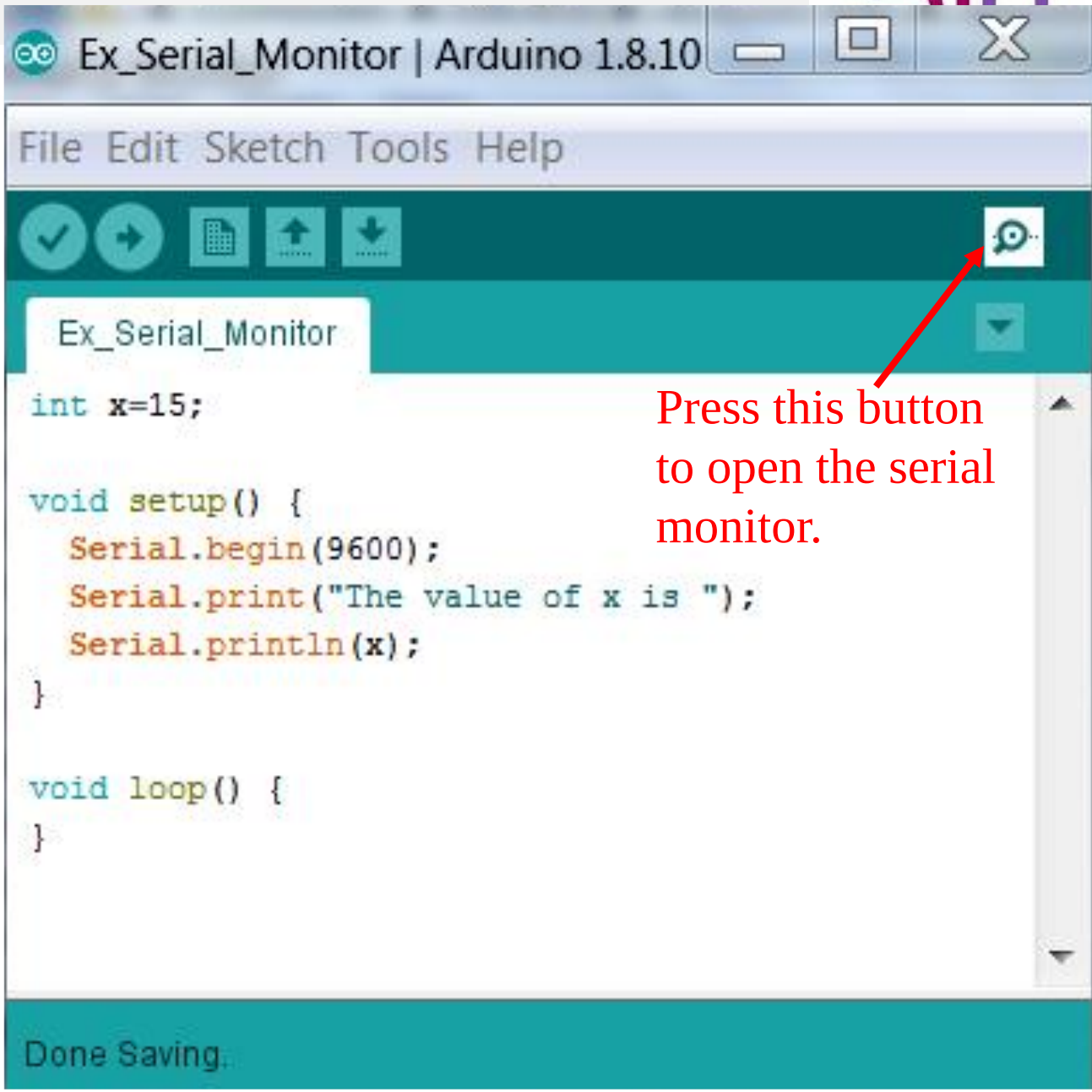
The following commands are used.

- ***Serial.begin(9600);*** → Initialize the serial monitor and set the baud rate to 9600 bps (default speed for the Arduino UNO). We must declare this command before we can use the serial monitor. Generally, we put this command in the ***setup()*** function.
- ***Serial.print("Message");*** → Print the text Message on the serial monitor.
- ***Serial.println("Message");*** → Print the text Message on the serial monitor and the cursor moves to the next line.
- ***Serial.print(x);*** → Print the value in the variable ***x*** on the serial monitor.
- ***Serial.println(x);*** → Print the value in the variable ***x*** on the serial monitor and the cursor moves to the next line.

Example: Using Serial Monitor

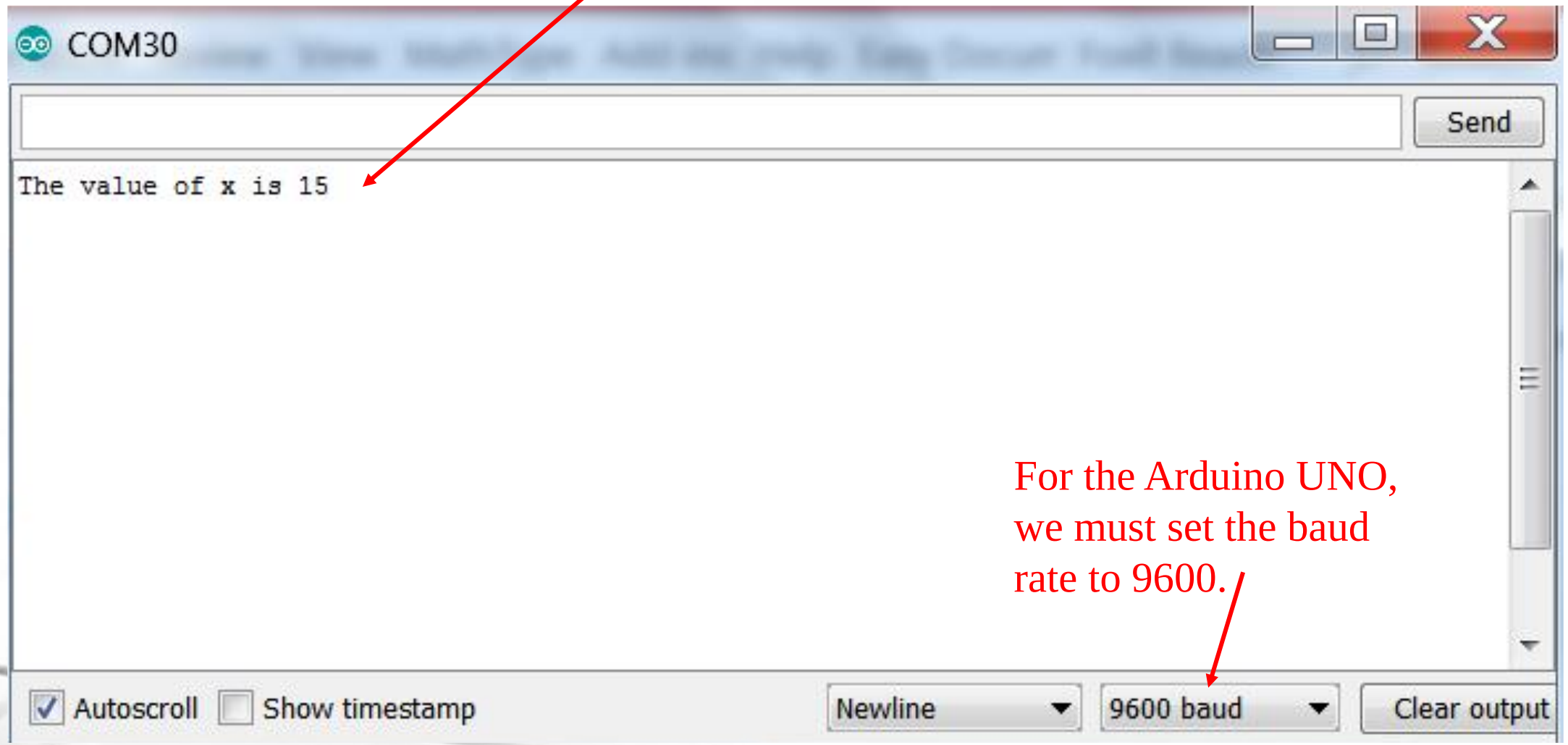
The following print the following text on the serial monitor:

The value of x is 15



Example: Using Serial Monitor

After upload this Arduino sketch to the Arduino UNO, when opening the serial monitor, we will see this windows and the text.



4) Creating a Time Delay

The following commands are used to create a time delay.

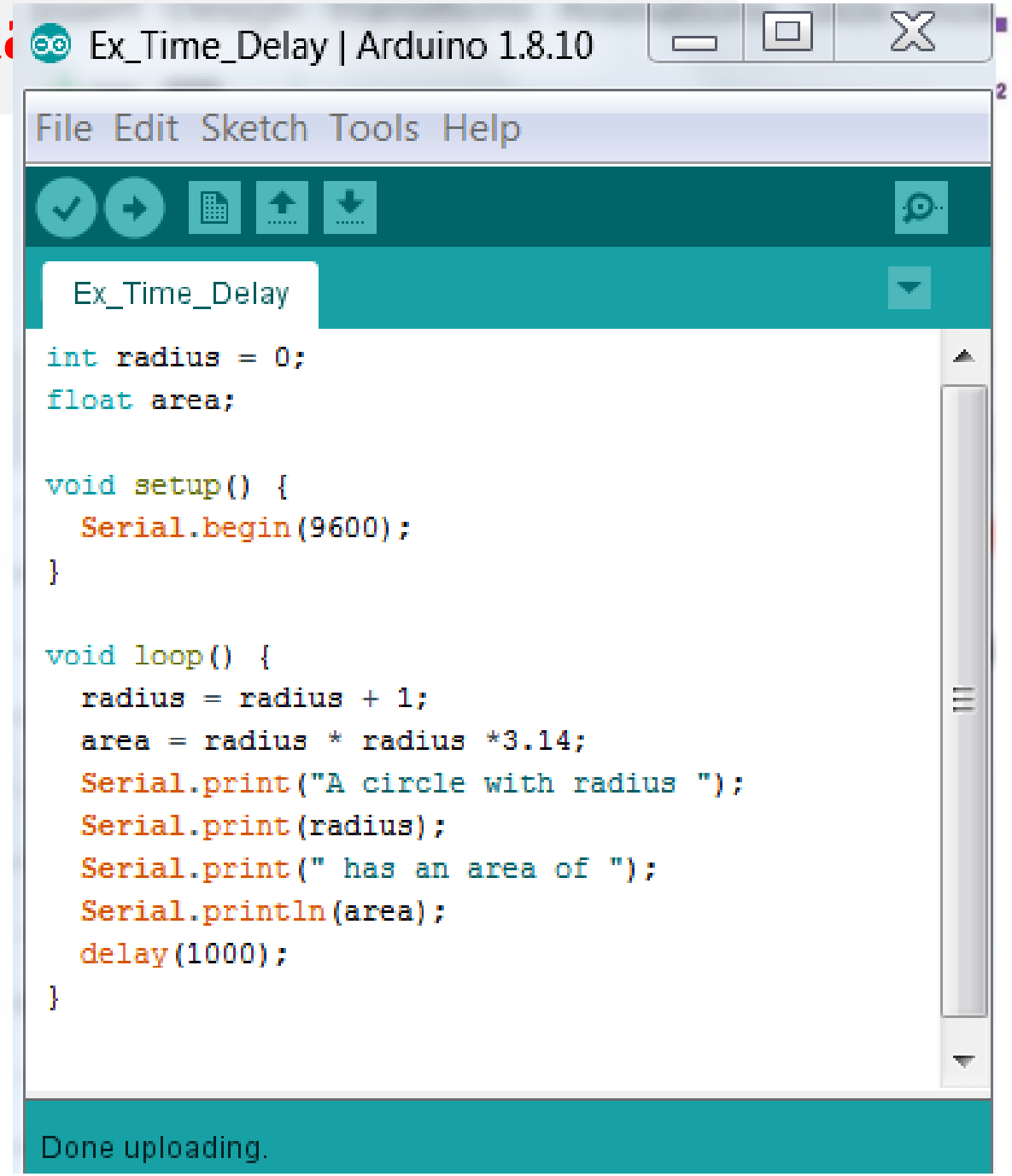
- ***delay(x);*** to create a time delay x milliseconds.
- ***delayMicroseconds(x);*** to create a time delay x microseconds.

Example:

- The command ***delay(1000);*** is to create a time delay 1000 milliseconds (or 1 second).
- The command ***delayMicroseconds(1000);*** is to create a time delay 1000 microseconds.

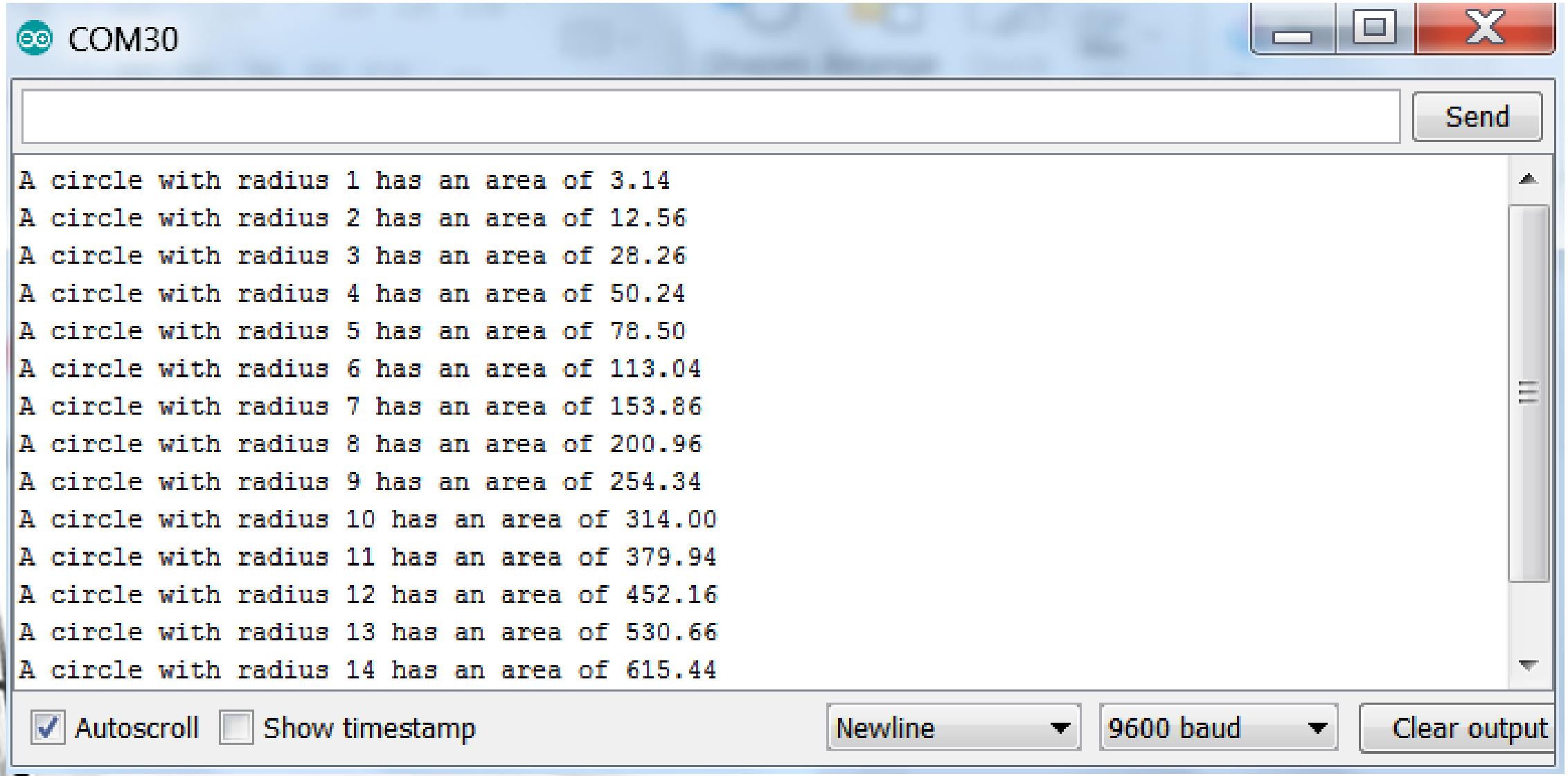
Example: Creating a Time Dela

Study the following Arduino sketch. What will be shown on the serial monitor?



Example: Creating a Time Delay

As a result, the following texts are shown on the serial monitor.



5) Working with Comparison Conditions

1. Numeric conditions: They are used to compare two variables and return TRUE or FALSE.

   Example: Assume that x=2 and y=10. The following conditions returns TRUE.
   - (x <= y)
   - (x != y)
   - (x < y)

Operator	Description
==	Equal
!=	Not equal
<>	Not equal
>	Greater than
>=	Greater than or equal
>	Less than
<=	Less than or equal

5) Working with Comparison Conditions

2. Compound conditions: We combine two or more conditions together by using **&&** or **||**.
   Example:
   - (x == 1) && (y > 5)
   - (x >= -10) || (y !=0)

3. Boolean conditions: When we use a variable as a condition, any variable whose value is not zero is recognized as TRUE; otherwise, it is recognized as FALSE.

6) Using the if, if-else, and else if Commands

1. The ***if*** command has the following format.
   - When the condition is true, these commands will be executed.

```
if (condition) {
    command;
    ...
    command;
}
```

2. The ***if-else*** command has the following format:
   - When the condition is true, these commands will be executed.
   - When the condition is false, these commands will be executed.

```
if (condition) {
    command;
    ...
    command;
} else {
    command;
    ...
    command;
}
```

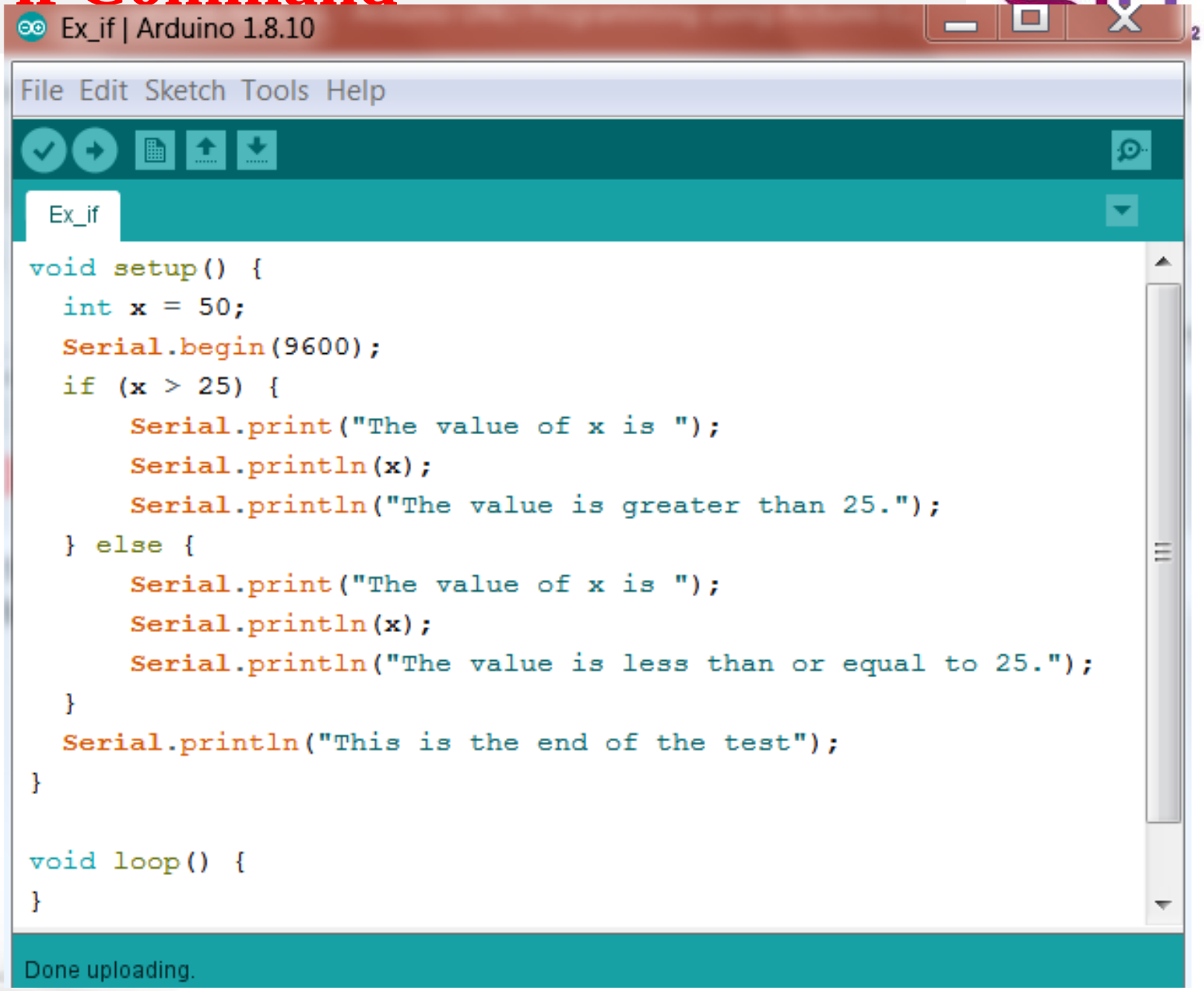
6) Using the if, if-else, and else if Command

3. The ***if*** and ***else if*** command has the following format (we apply this command when we have many conditions):
   - When the condition1 is true, these commands are executed.
   - When the condition2 is true, these commands are executed.
   - When the condition3 is true, these commands are executed.
   - When the no conditions are true, these commands are executed.

```
if (condition1) {
    command;
    ...
    command;
} else if (condition2) {
    command;
    ...
    command;
} else if (condition3) {
    command;
    ...
    command;
} else {
    command;
    ...
    command;
}
```

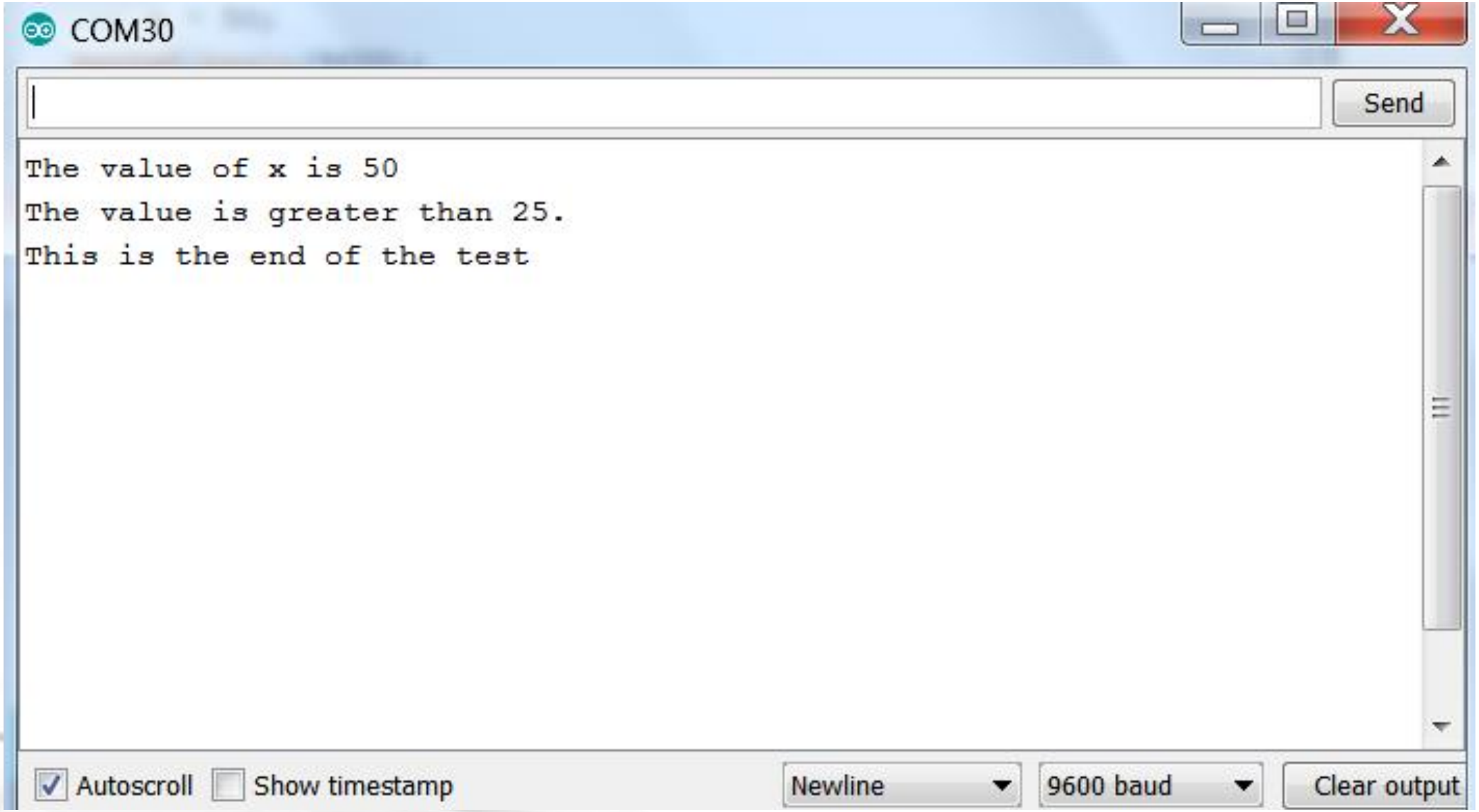
Example: Using the if Command

Study the following sketch. What will be shown on the serial monitor?



Example: Using the if Command

As a result, the following texts are shown on the serial monitor.



7) Using the while and do-while Loops

1. The ***while*** loop has the following format.
   - Repeat the loop until the condition is false.

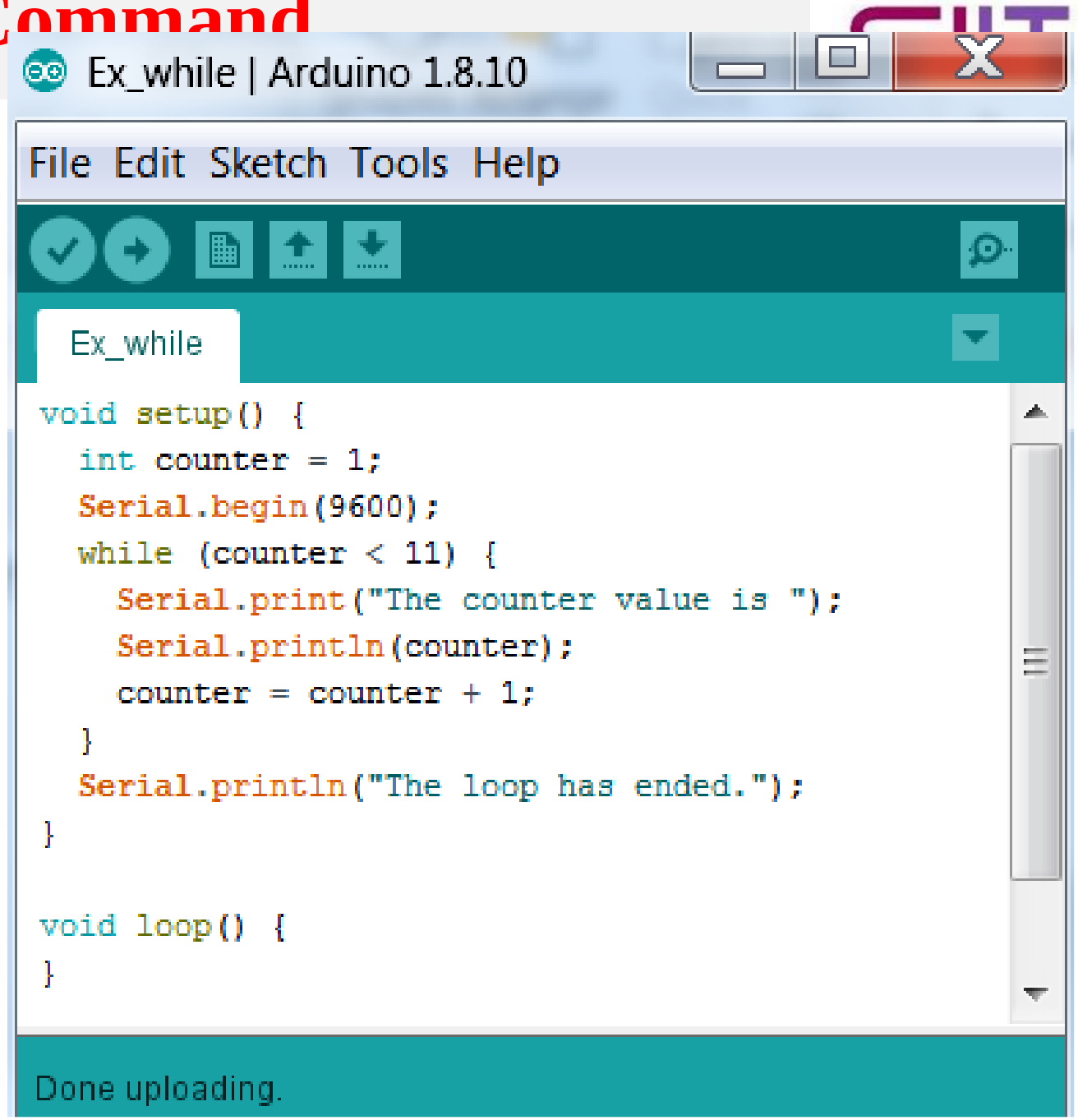
```
while (condition) {
    command;
    ...
    command;
}
```

2. The ***do-while*** loop has the following format.
   - Here, when entering the do-while loop, first time, the condition is not checked, the Arduino will execute these commands.
   - Thereafter, it will check the condition. It repeats the loop until the condition is false.

```
do {
    command;
    ...
    command;
} while (condition);
```

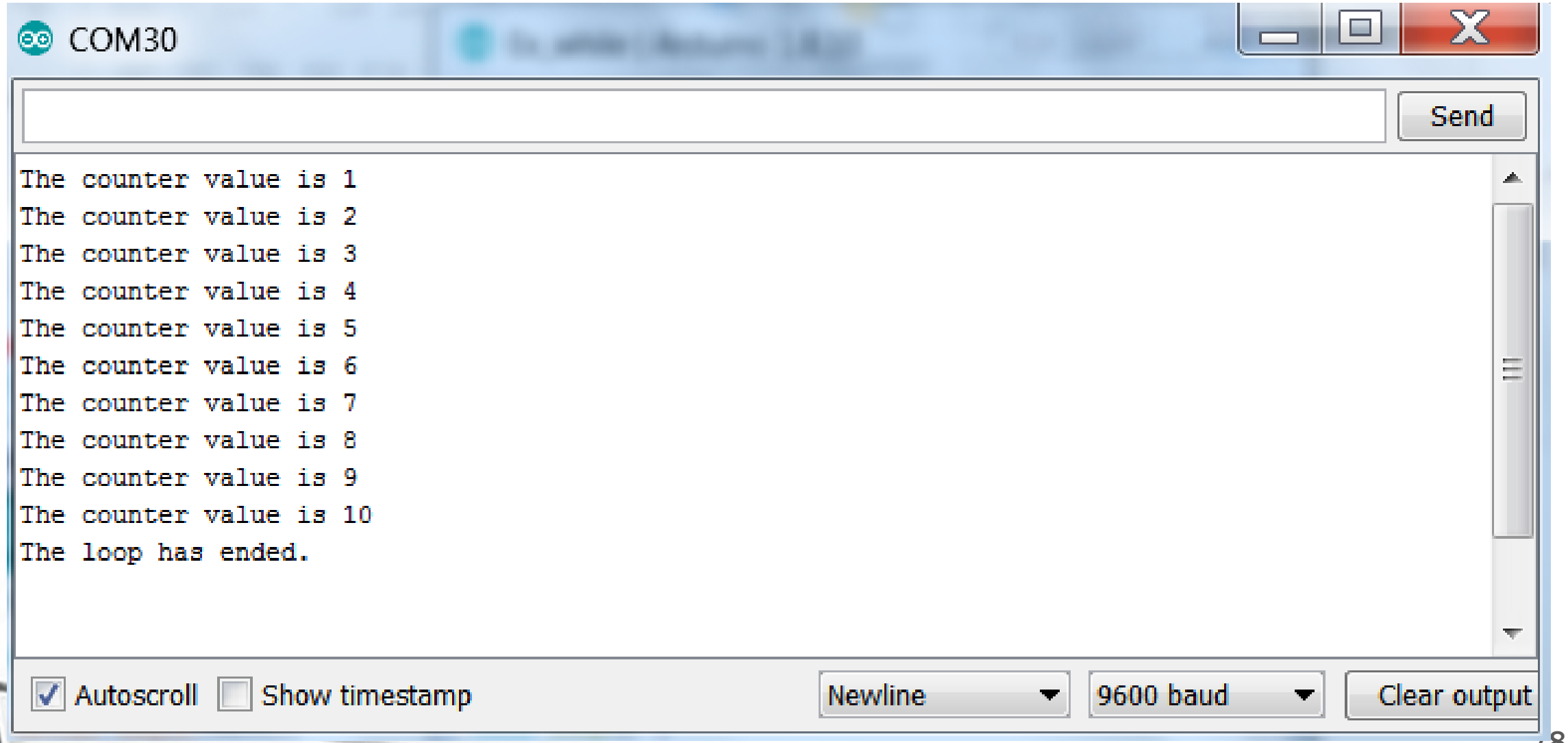
Example: Using the while Command

Consider the following sketch. What will be shown on the serial monitor?



Example: Using the while Command

As a result, the following texts are shown on the serial monitor.



8) Using the for Loop

The ***for*** loop has the following format:

- statement1 is to set up an initial value of the counter variable.
- condition indicates when to stop the for loop.
- statement2 indicates how to update the counter variable.

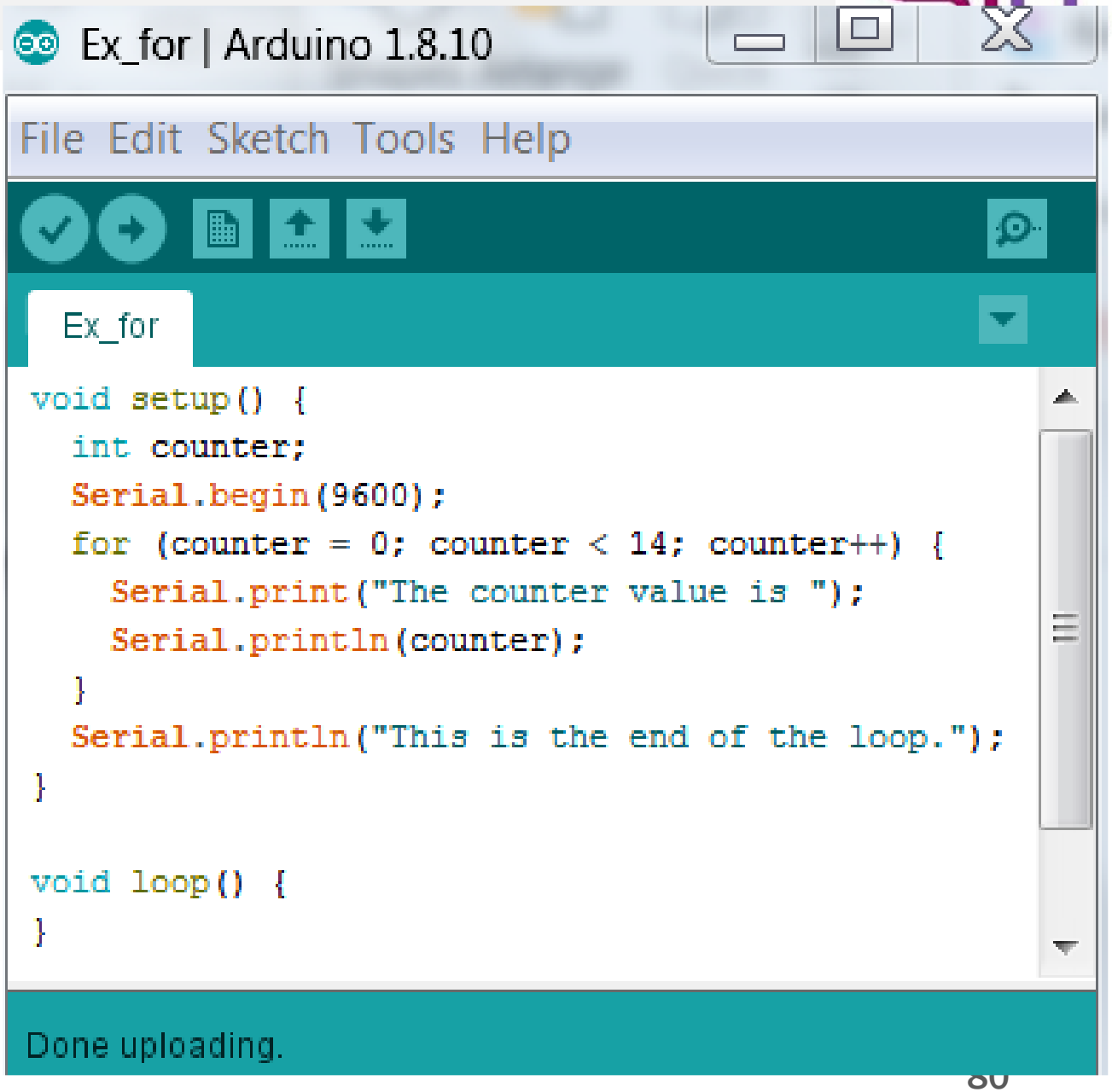
```
for (statement1; condition; statment2) {
        command;
        ...
        command;
}
```

Example: Here is to repeat the loop for 15 times

```
for (x=1; x<=15; x++) {
        command;
        ...
        command;
}
```

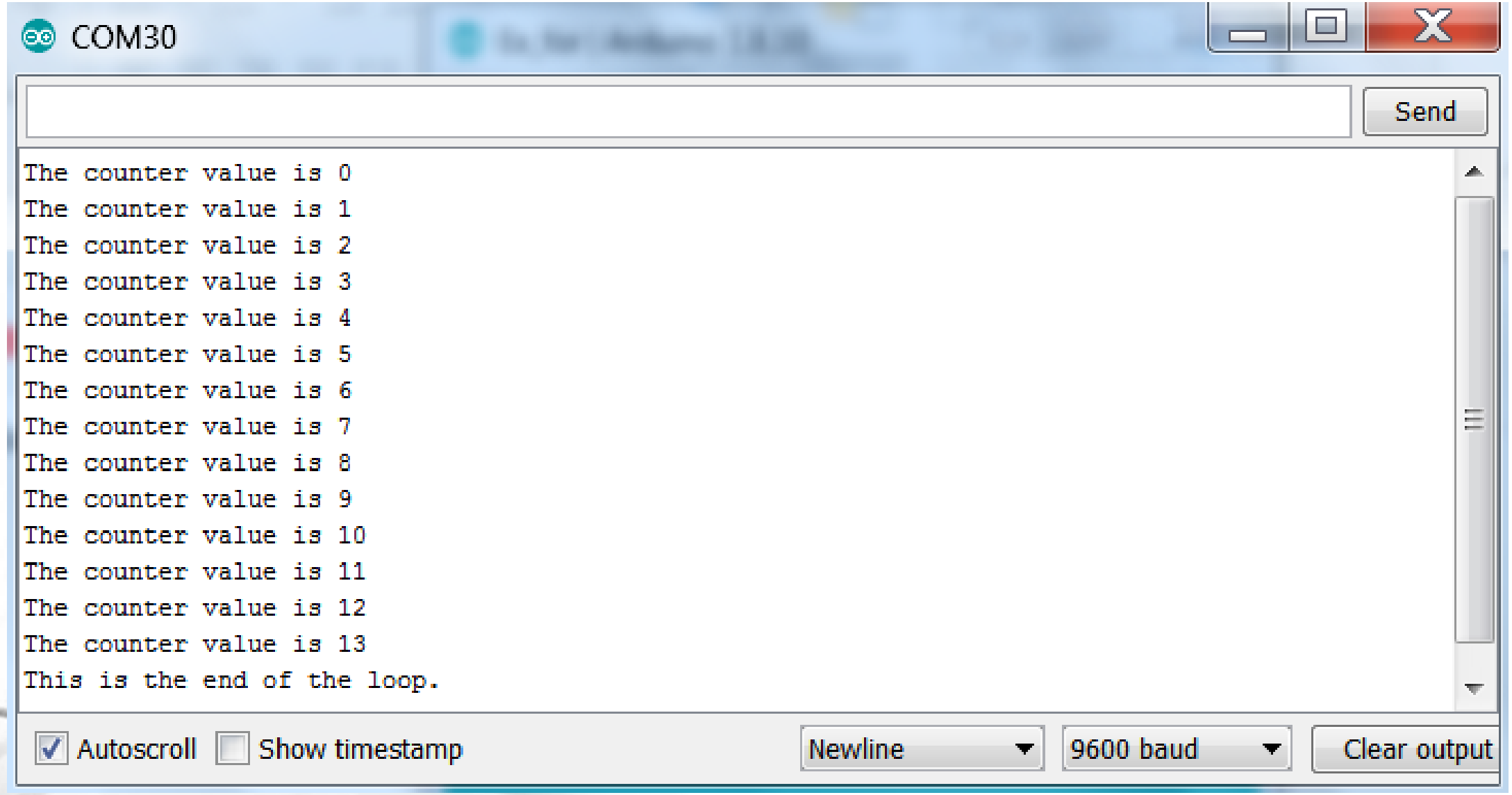
Example: Using the for Command

Consider the following sketch. What will be shown on the serial monitor?



Example: Using the for Command

As a result, the following texts are shown on the serial monitor.



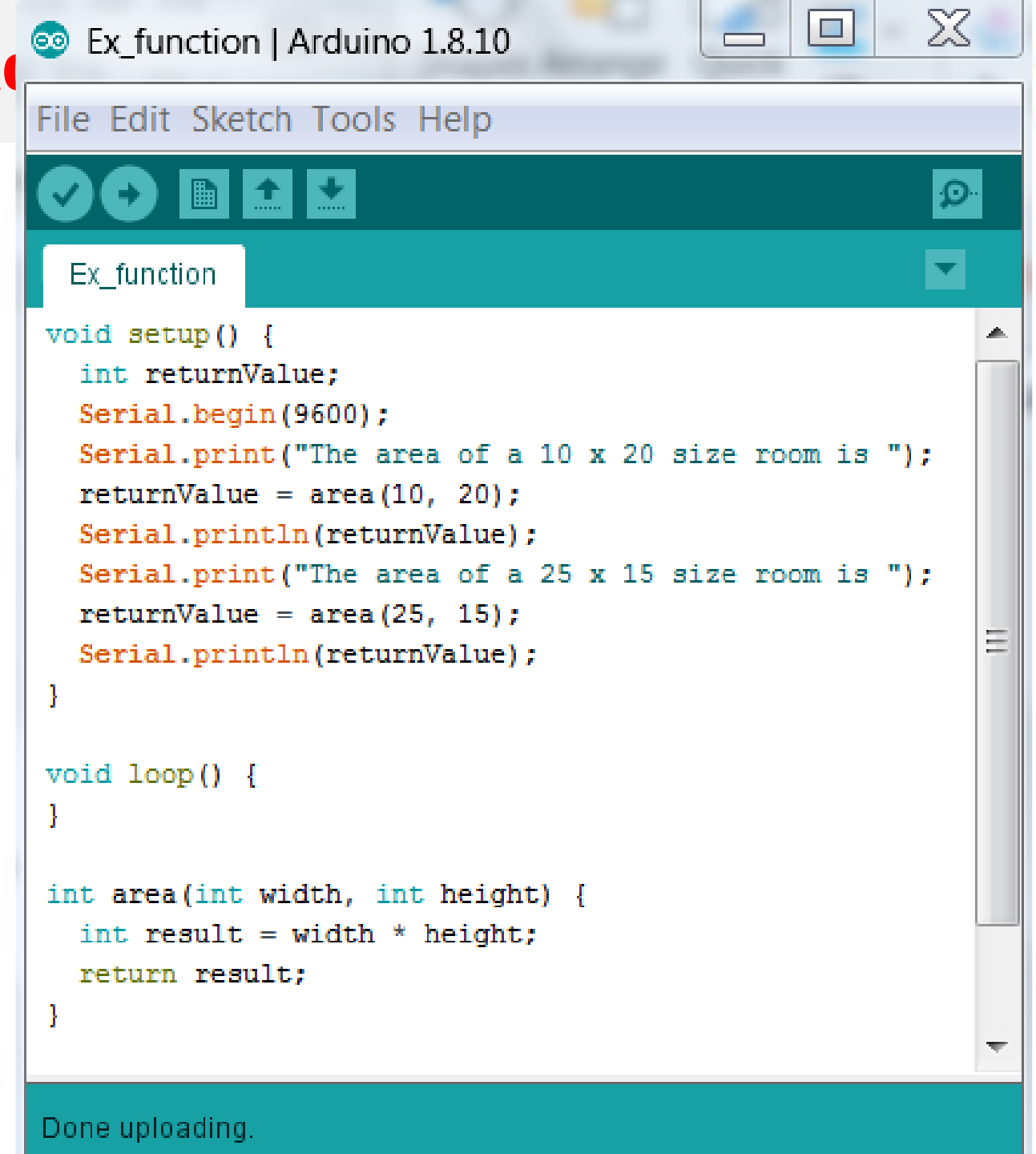
9) Creating a Function

- When should we create a function?
  - When we have to use a set of commands again and again, we should group them inside a function.
- A function has the following format:
  - The ***func_name*** is the name of this function. We will call this name whenever we uses it.
  - The ***datatype*** defines the data type of the value returned by this function.
  - The ***input_para*** defines the input parameter to this function (can be more than one input).

```
datatype  func_name (input_para) {
    command;
    ...
    command;
}
```

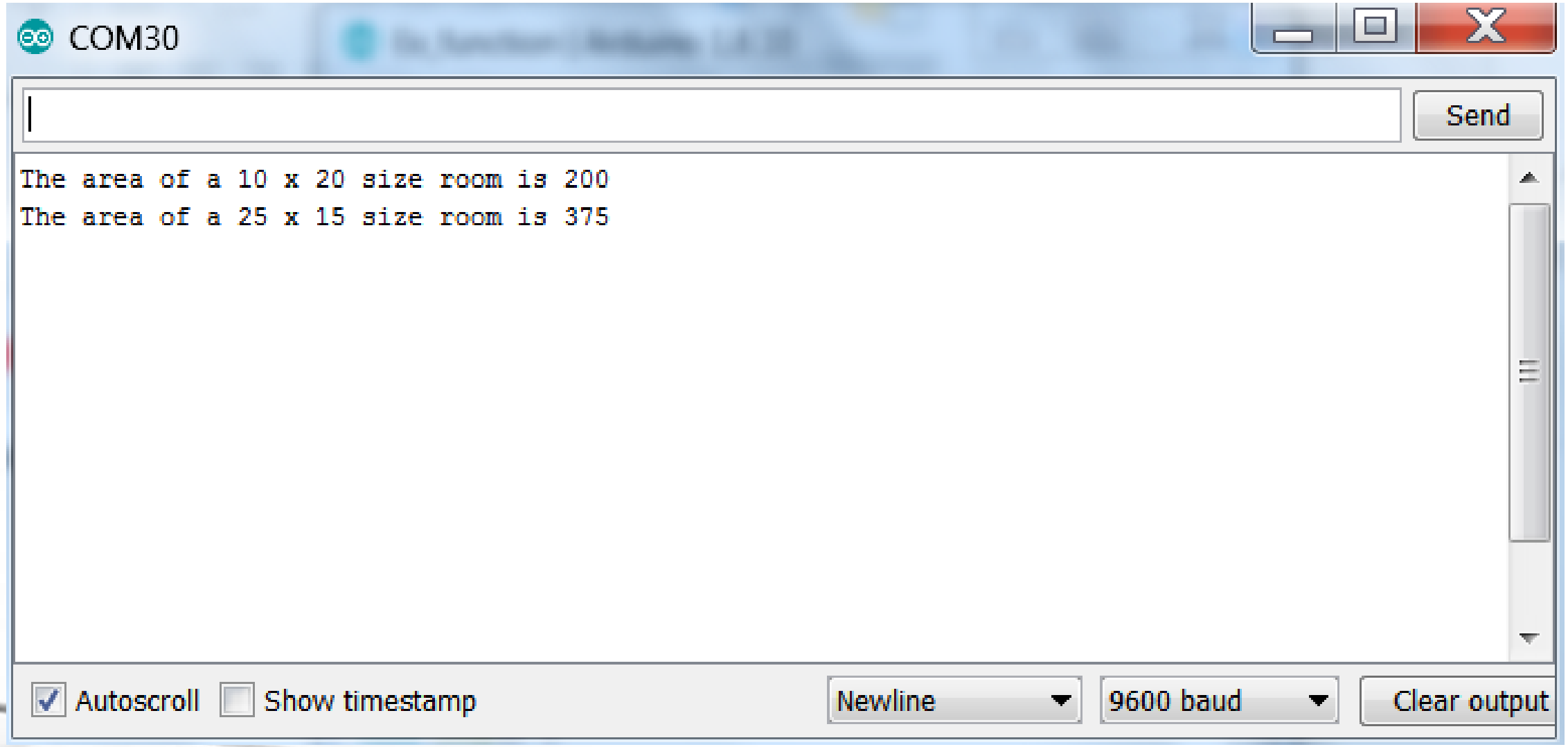
Example: Creating a Functi

Consider the following sketch. What will be shown on the serial monitor?



Example: Creating a Function

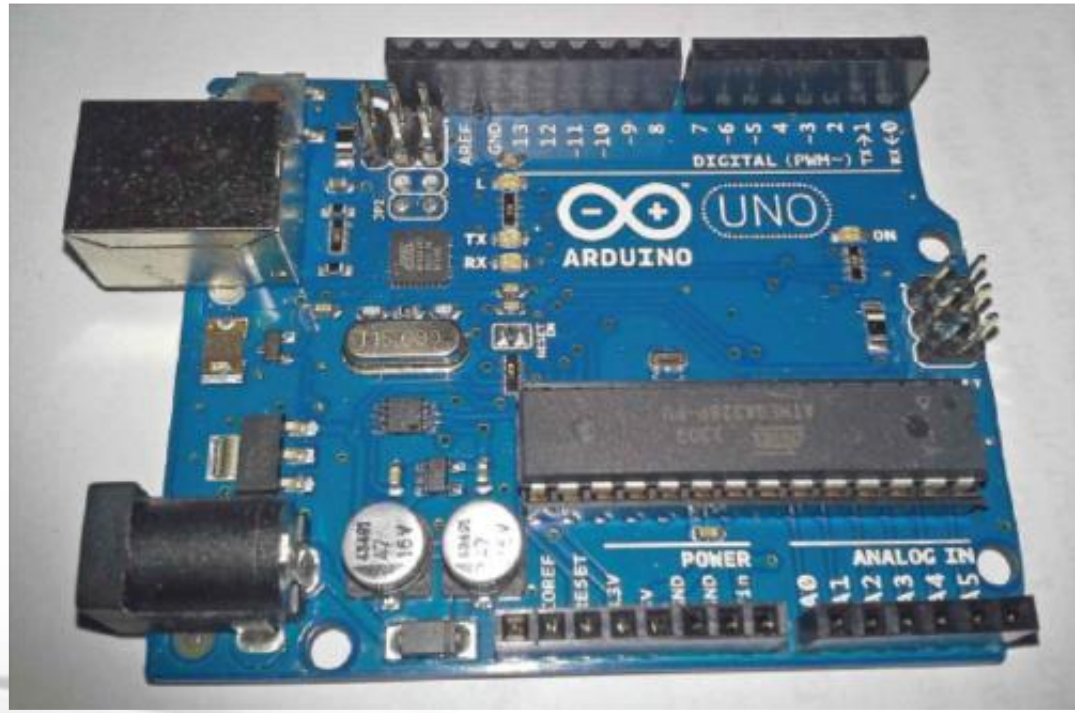
As a result, the following texts are shown on the serial monitor.



Digital Interfaces/Pins

Digital Interface Overview

- The primary purpose of the Arduino UNO is to control external devices. Two types of Arduino Interfaces are digital interfaces (= pins) and analog interfaces (= pins).
- There are 14 digital pins in the Arduino UNO.
  - The pins with the labels 0 – 13.
  - Each pin can be an input pin (to read) or an output pin (to write).
  - The value through each pin can be only 0 (LOW) or 1 (HIGH).



Digital Pin's Commands

There are three Arduino commands to control a digital pin:

- ***pinMode(pin, MODE);***
- ***digitalWrite(pin, value);***
- ***digitalRead(pin);***

Setting the Pin Mode

A digital pin can be set as an input pin (to read) or output pin (to write). We can set up that pin by using the command:

pinMode(pin, MODE);

- The parameter ***pin*** is a number between ***0*** and ***13*** to indicate a pin between 0 – 13.
- The parameter ***MODE*** is to indicate whether this pin is an input or an output. We use the term ***INPUT*** to set it as an input pin. We use the term ***OUTPUT*** to set it as an output pin.

Example:

- The command ***pinMode(4, INPUT);*** is used to set the digital pin 4 to be an input pin.
- The command ***pinMode(7, OUTPUT);*** is used to set the digital pin 7 to be an output pin.

Write an Output to a Digital Pin

We can send a value 0 (LOW) or 1 (HIGH) to a digital pin if that pin has already set as an output pin. We use the following command:

digitalWrite(pin, value);

- The parameter ***pin*** is a number between *0* and *13* to indicate a pin between 0 – 13.
- The parameter ***value*** is 0 or 1 to indicate LOW or HIGH.

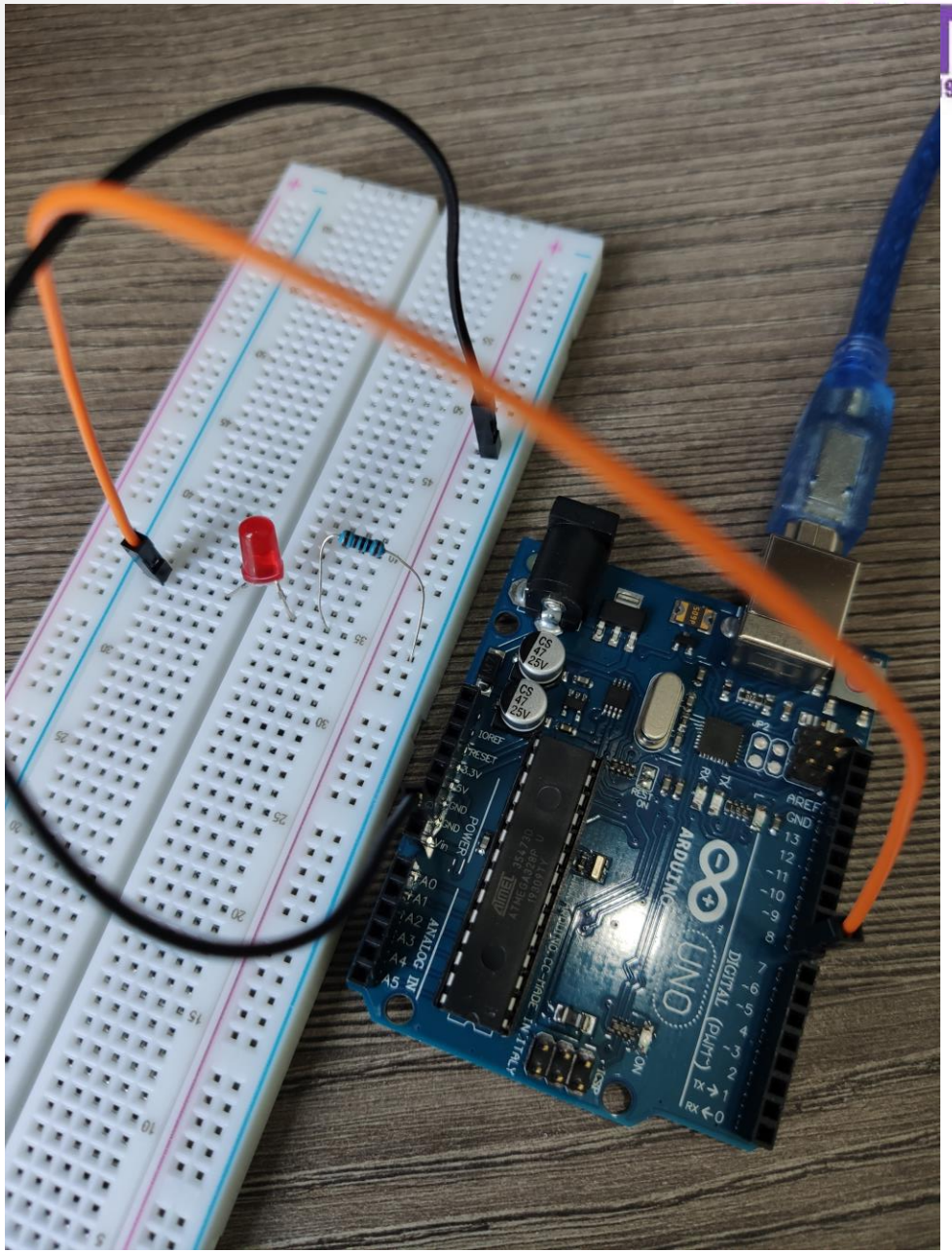
Note that when a pin is set up as an output pin, this pin will supply a voltage: 0 indicates LOW (around 0 volt) and 1 indicates HIGH (around 5 volt).

Example:

- The command ***digitalWrite(4, 1);*** is used to supply around 5 volt through the pin 4.
- The command ***digitalWrite(7, 0);*** is used to supply around 0 volt through the pin 7.

Example: Blink an LED

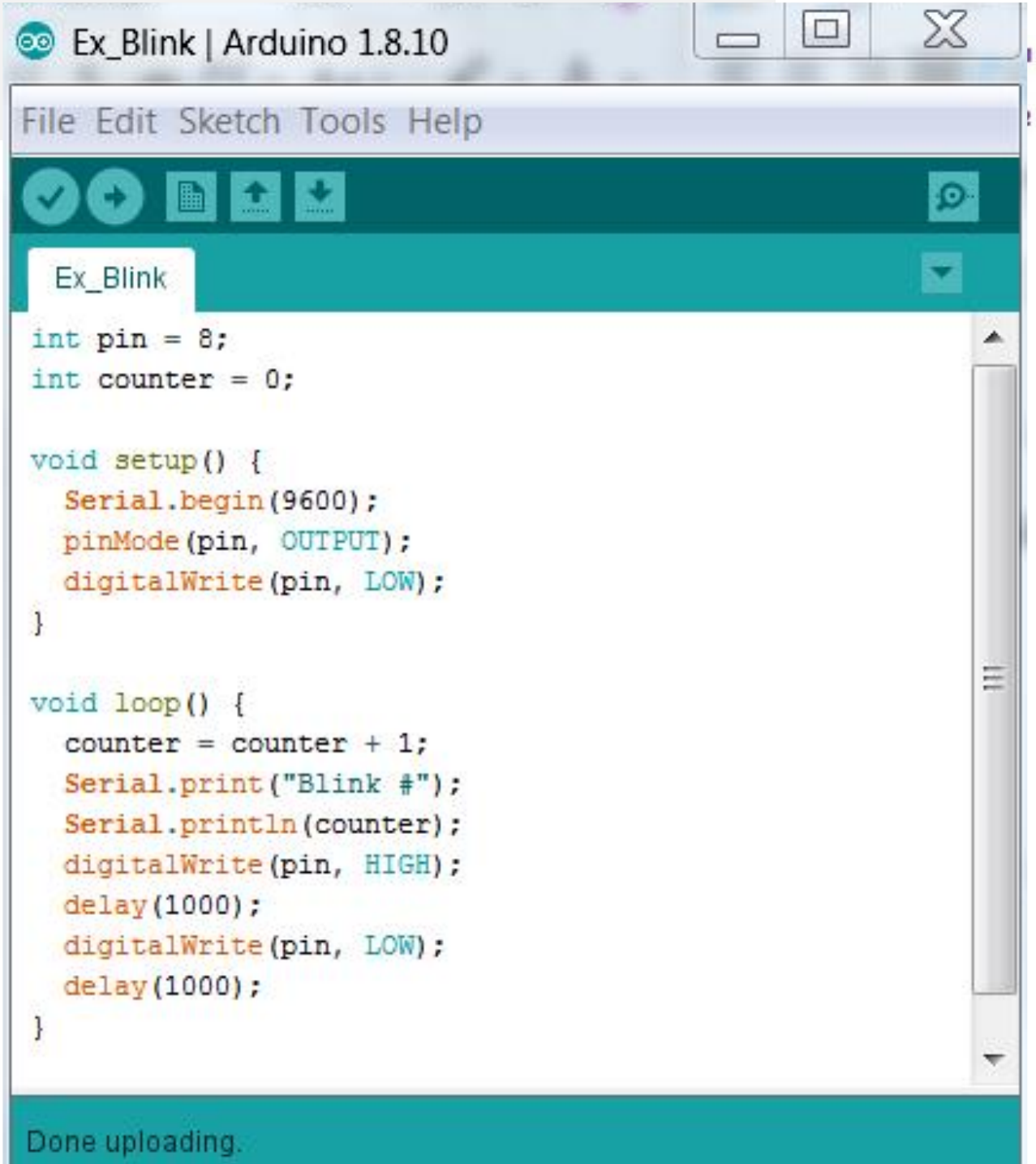
We connect an LED on the digital pin 8 of the Arduino UNO as shown below.



Example: Blink an LED

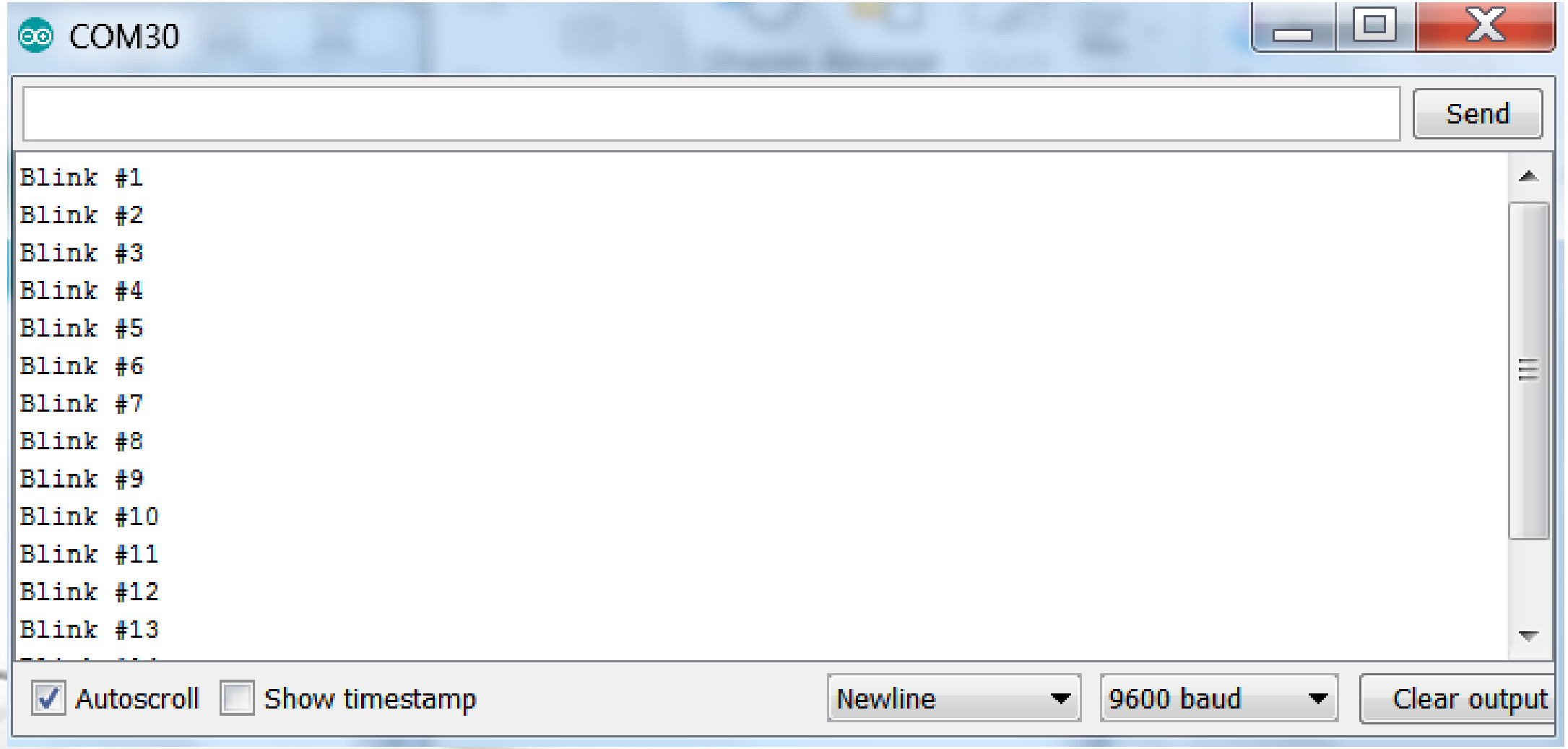
The following Arduino sketch will:

- print the number of blinks on the serial monitor,
- turn the LED on for 1 second,
- turn the LED off for 1 second.



Example: Blink an LED

As a result, we will see that the LED will be on for 1 second and off for 1 second. In addition, the serial monitor will print the following messages, every 2 seconds.



Read an Input from a Digital Pin

We can read a value 0 (LOW) or 1 (HIGH) from a digital pin if that pin has already set as an input pin. We use the following command:

X = digitalRead(pin);

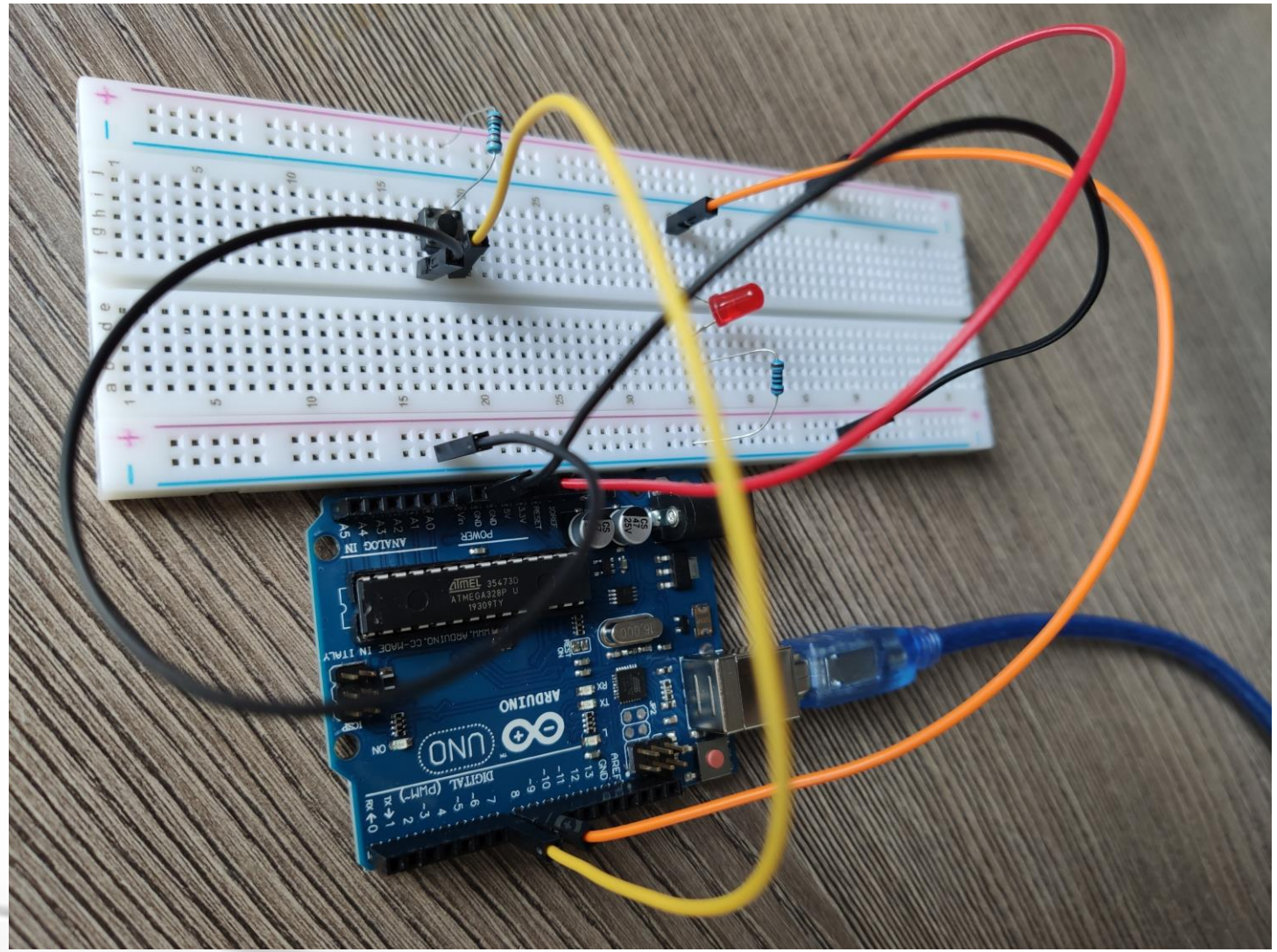
- The variable X (declared as an integer data type) is a dummy variable to store the value (0 or 1) which is returned from the command ***digitalRead(pin);***
- The parameter ***pin*** is a number between *0* and *13* to indicate a pin between 0 – 13.

Example:

- The command ***X=digitalRead(4);*** is used to read the digital value (0 or 1) from the pin 4 and store it in the variable ***X***.
- The command ***X=digitalRead(7);*** is used to read the digital value (0 or 1) from the pin 7 and store it in the variable ***X***.

Example: Read a Digital Value from a Digital Pin

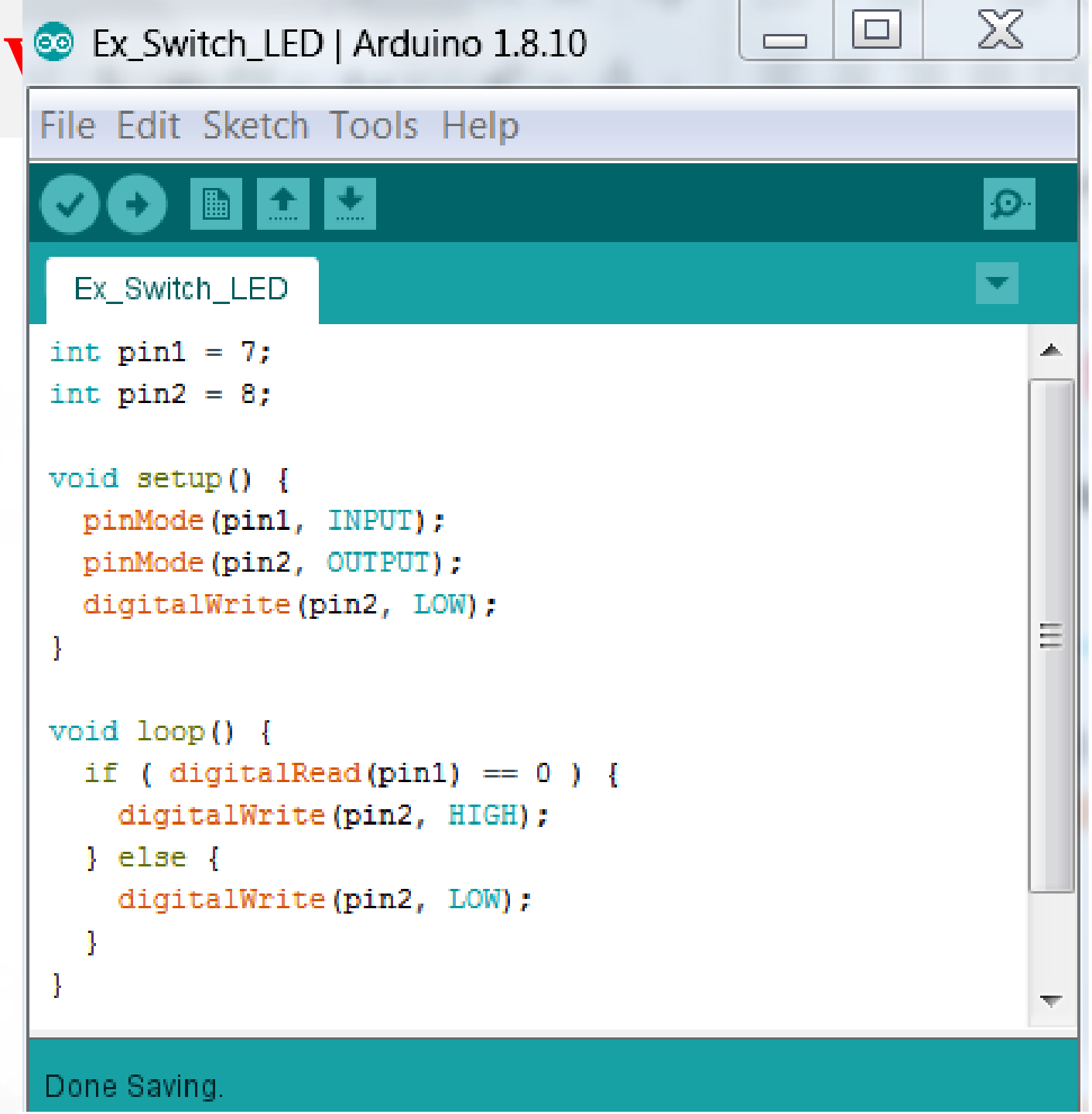
Connect a switch to the digital pin 7 and an LED to the digital pin 8 as shown below.



Example: Read a Digital V

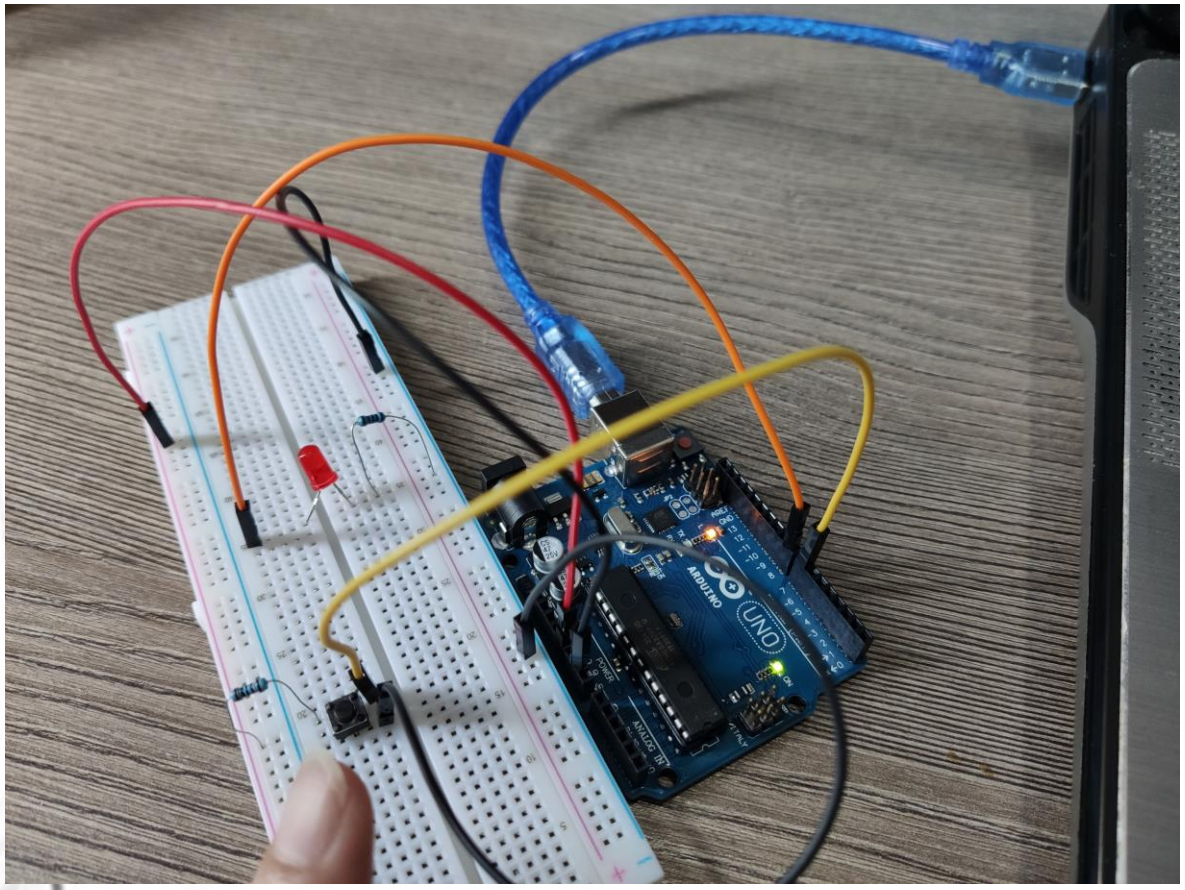
The following Arduino sketch will:

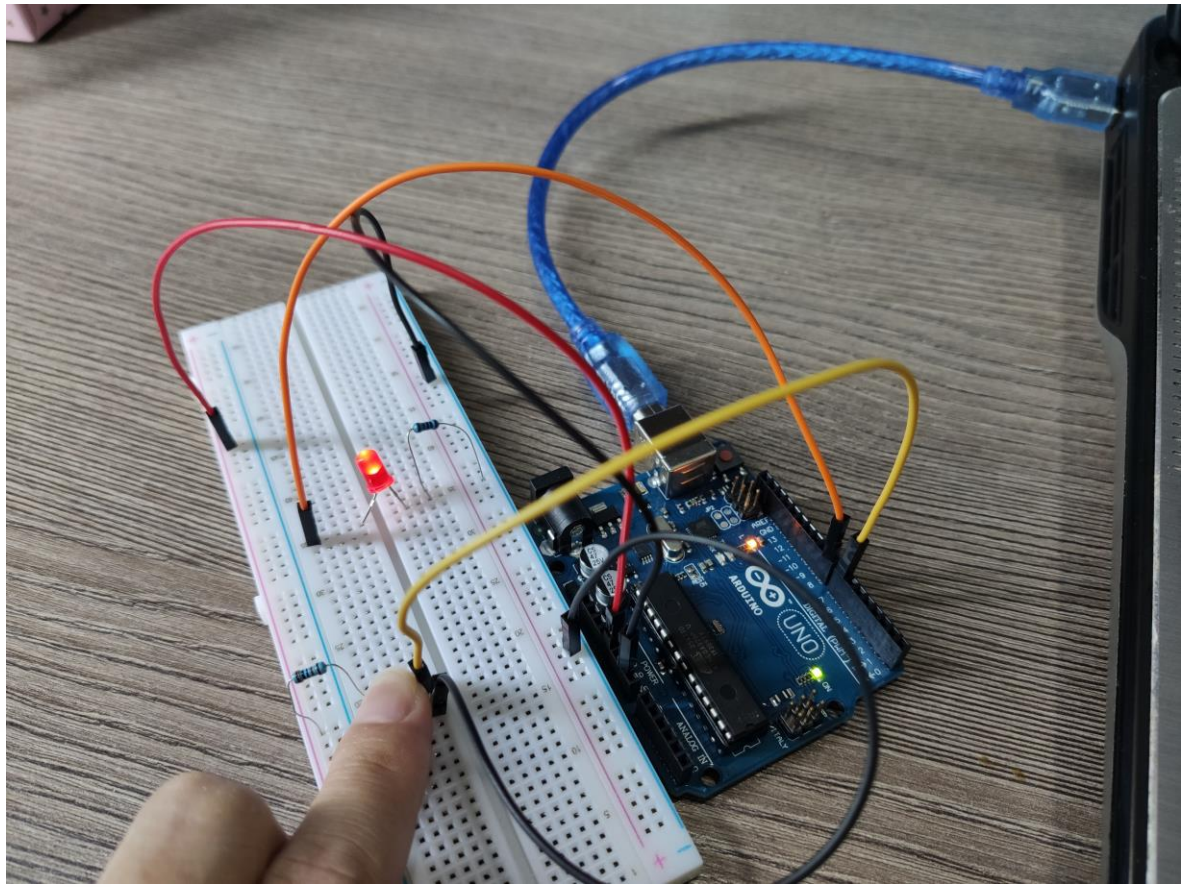
- turn the LED on if we press the switch,
- turn the LED off if we not press the switch.



Example: Read a Digital Value from a Digital Pin

As a result, when we press the switch, the LED is on; otherwise, the LED is off.

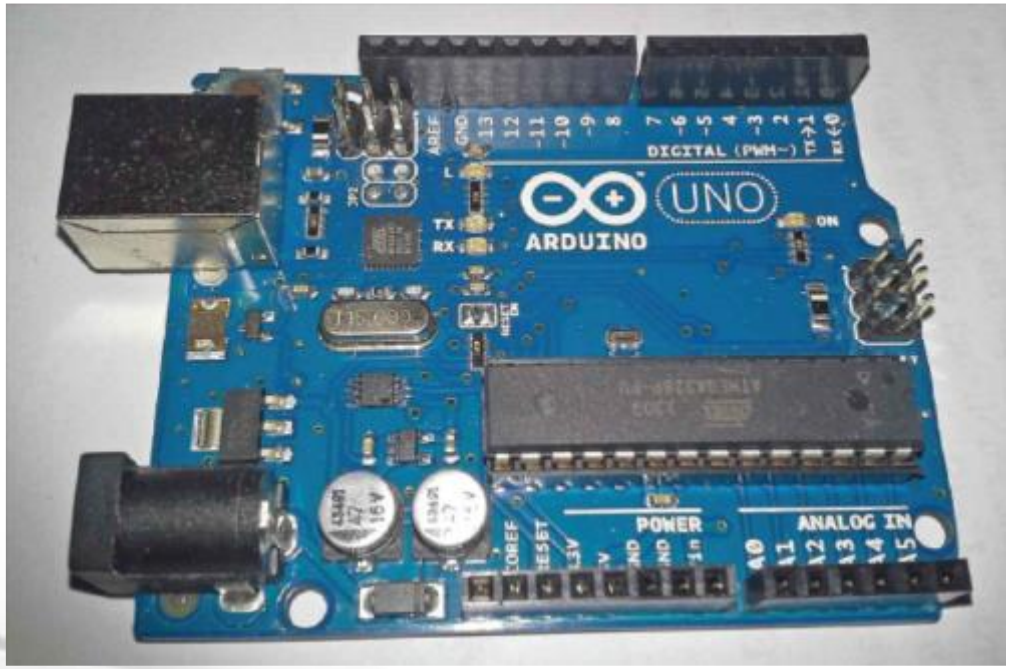




Analog Interfaces/Pins

Analog Interface Overview

- Microcontrollers are built from digital electronics. However, they can interact with analog devices.
- The following pins are used to interact with input/output analog devices.
  - The pins A0 – A5 are used to connect with input analog devices (= read data from these devices). → Analog Pins
  - The digital pins 3, 5. 6, 9, 10, and 11 (pins with the ~ sign, which is the PWM pins) are used to connect with output analog device (= control these devices). → PWM Pins



Analog and PWM Pins' Commands

There are two Arduino commands to interact with these pins:

- For a PWM pin: ***analogWrite(pin, value);***
- For an analog pin: ***analogRead(pin);***

Note that, we do not need to set the pin modes of these pins when using them.

Write a Voltage Value to a PWM Pin

We can send an integer number between 0 and 255 to a PWM pin, where 0 is equal to 0V, 255 is equal to 5V, and the other number maps to a voltage proportionally. We use the following command:

analogWrite(pin, value);

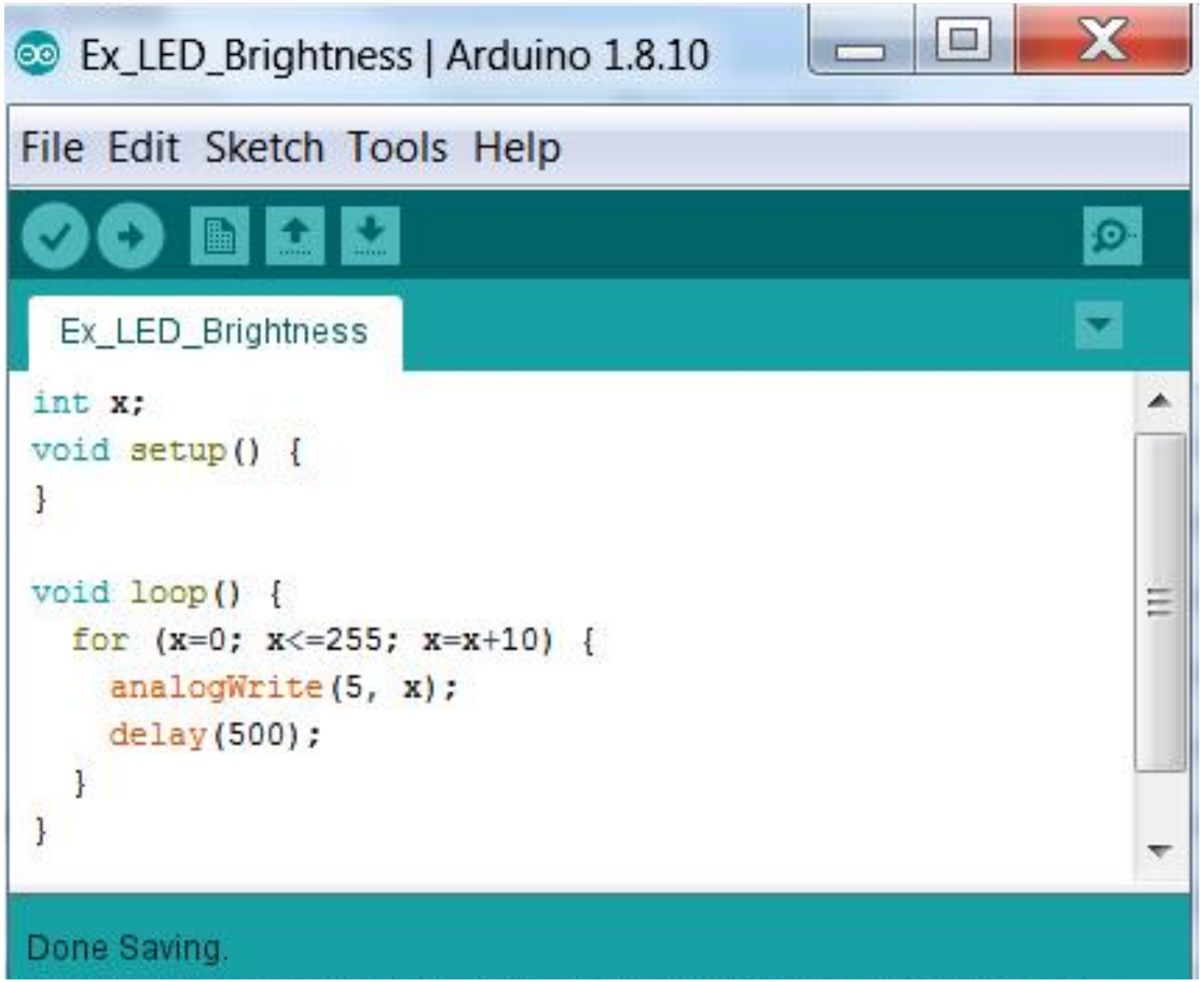
- The parameter ***pin*** is the pin name which is 3, 5, 6, 9, 10, or 11 to indicate the PWM pin that we is using.
- The parameter ***value*** is between 0 and 255.

Example:

- The command ***analogWrite(5, 0);*** is used to supply around 0 volt through the PWM pin 5.
- The command ***analogWrite(10, 200);*** is used to supply around $\frac{200\times5}{255} = 3.92\text{V}$ to the PWM pin 10.

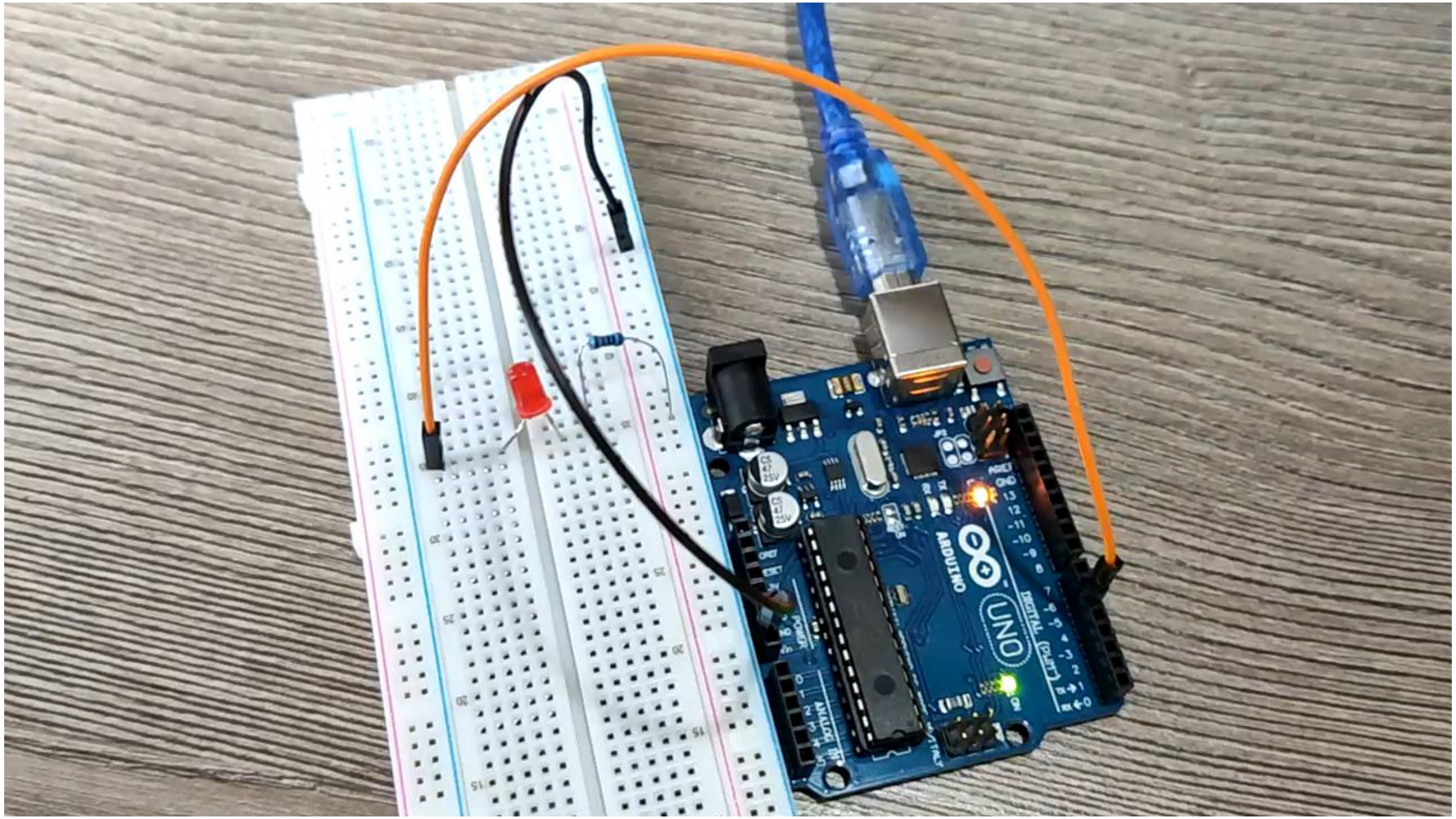
Example: Write a Value to a PWM Pin

Connect an LED to the digital pin 5 (which is also a PWM pin). The following Arduino sketch increases the brightness of the LED.



Example: Write a Value to a PWM Pin

As a result, the brightness/intensity of the LED is increased.



Read an Input Voltage from an Analog Pin

We can read an input voltage from an analog pin. This input voltage will be converted to a 10-bit binary number by an analog-to-digital converter (ADC), where we will see this number as an integer number between 0 and 1023.

- An input voltage (0V – 5V) will proportionally mapped to 0 – 1023, where 0V is converted to the number 0 and 5V is converted to the number 1023.

We use the following command:

X = analogRead(pin);

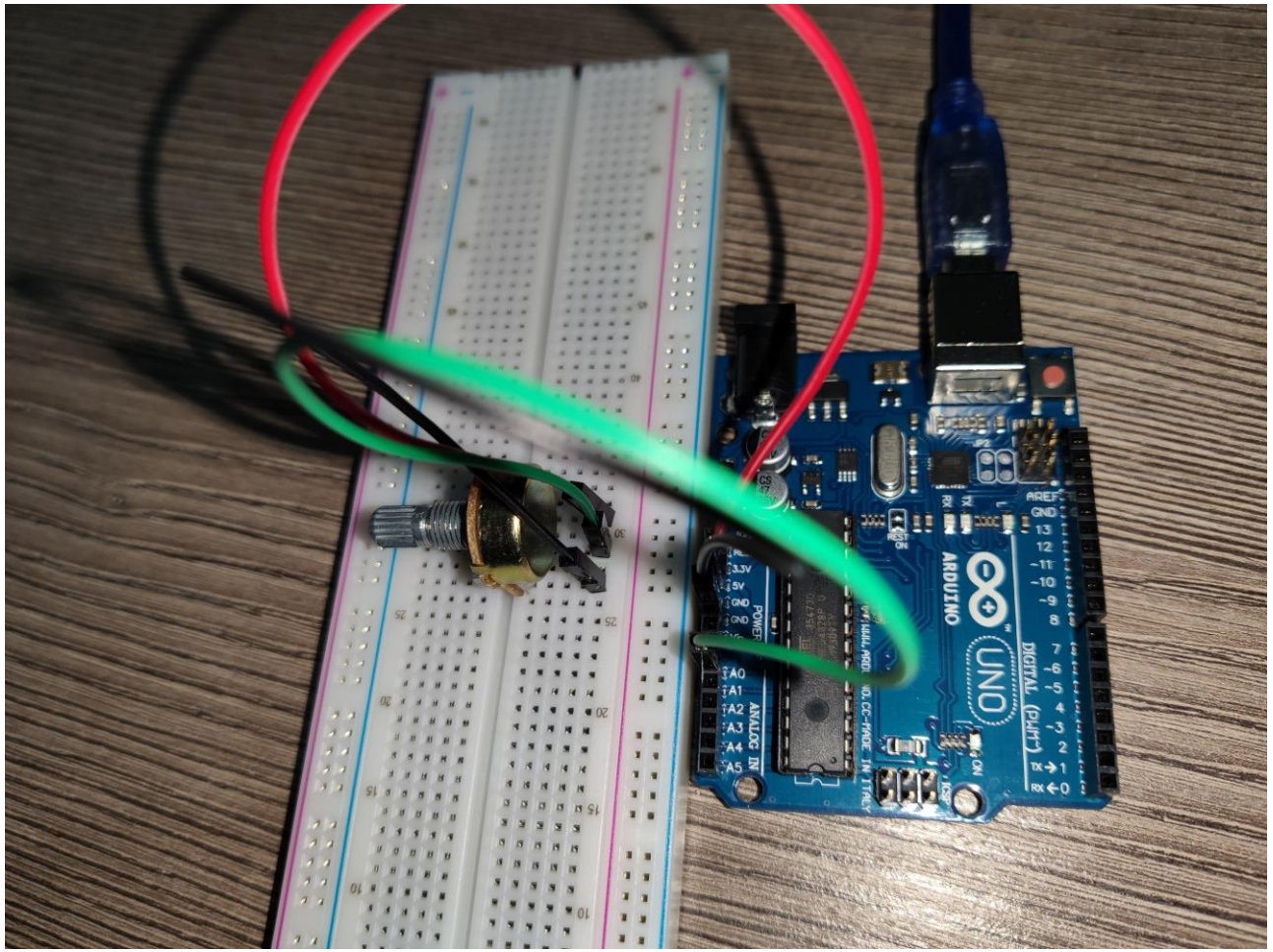
- The variable ***X*** (declared as an integer data type) is a dummy variable to store the value (0 – 1023) which is returned from the command ***analogRead(pin);***
- The parameter ***pin*** is an analog pin between ***A0*** and ***A5***.

Example:

- The command ***X=analogRead(A0);*** is used to read the input voltage from the pin ***A0*** and store it in the variable ***X***.
- The command ***X=analogRead(A4);*** is used to read the input voltage from the pin ***A4*** and store it in the variable ***X***.

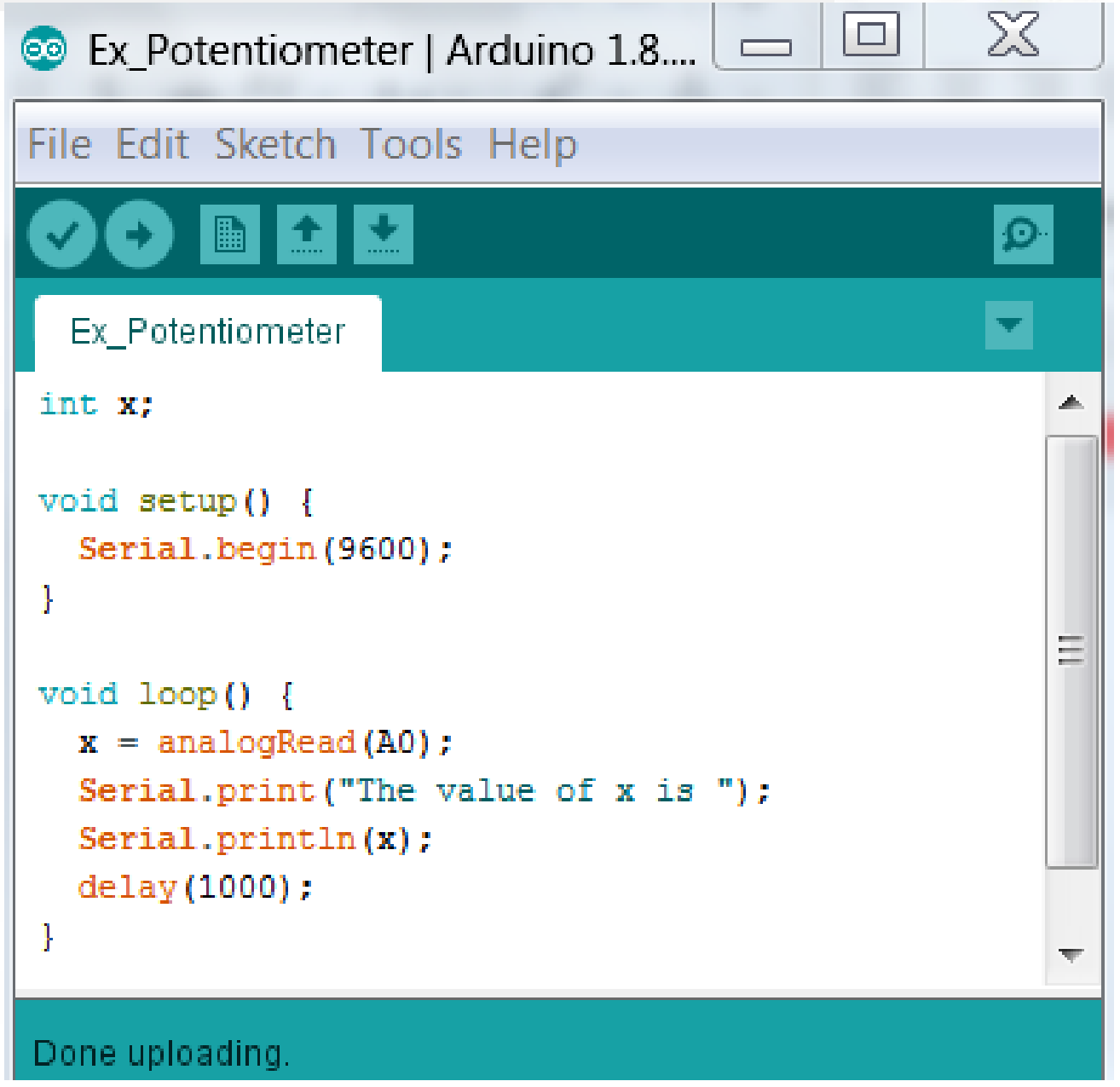
Example: Read a Value from an Analog Pin

Connect a potentiometer to the analog pin A0.



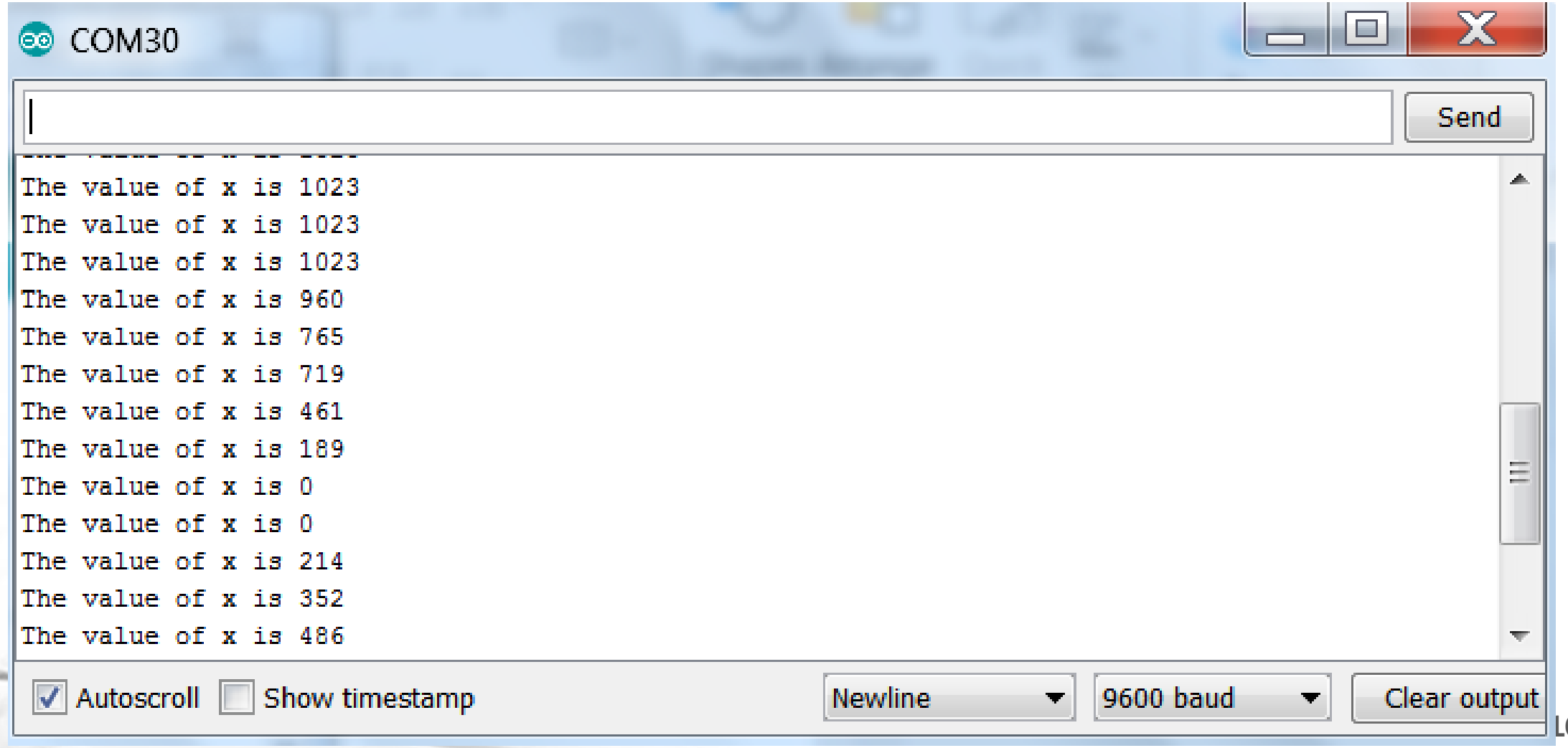
Example: Read a Value from an Analog Pin

The following Arduino sketch will read the input value at the analog pin A0 and print it on the serial monitor, every 1 second.



Example: Read a Value from an Analog Pin

As a result, while rotating the knob of the potentiometer (changing the resistance), we will see the following text on the serial monitor.



3. Examples of Sensors and Actuators

Sensors

A *sensor* is a device (connected to a microcontroller) that is used to sense an environment (such as temperature, gas, light intensity, etc.) and send this value to the microcontroller. It is an input device.

Actuators

An *actuator* is a device (connected to a microcontroller). The microcontroller controls this device to do an action such as alarm, movement, etc. It is an output device.

รหัสสินค้า AG00021
DC motor 1.5-9 Volt

รหัสสินค้า AG00005
Metal Gear Servo MG90S (เฟืองทำจากโลหะ)

รหัสสินค้า SG10157
1602 Module 16x2 LCD Display + I2C Interface

รหัสสินค้า AG10224
RGB OLED 0.95" 96x64 Pixels 16-bit Color OLED -

รหัสสินค้า RU00044
Control Glove ถุงมือกลควบคุม

Necessary Information of Using Sensors/Actuators

When we are using a sensor or actuator, the following information are necessary:

1. Circuit connection. → we need to know how to connect this sensor/actuator to our microcontroller correctly.
2. An example of code using this sensor/actuator. → we apply this code to our application.

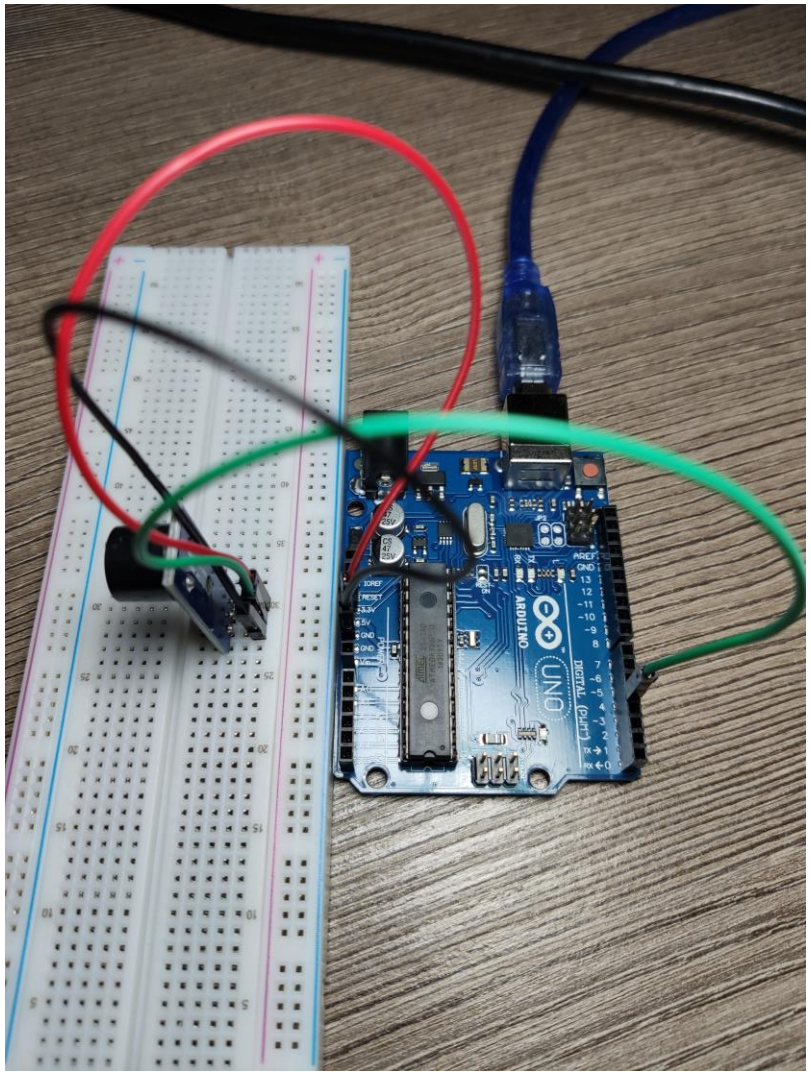
Buzzer

A buzzer is an actuator that is used to make a sound/alarm.

1. Circuit Connection:

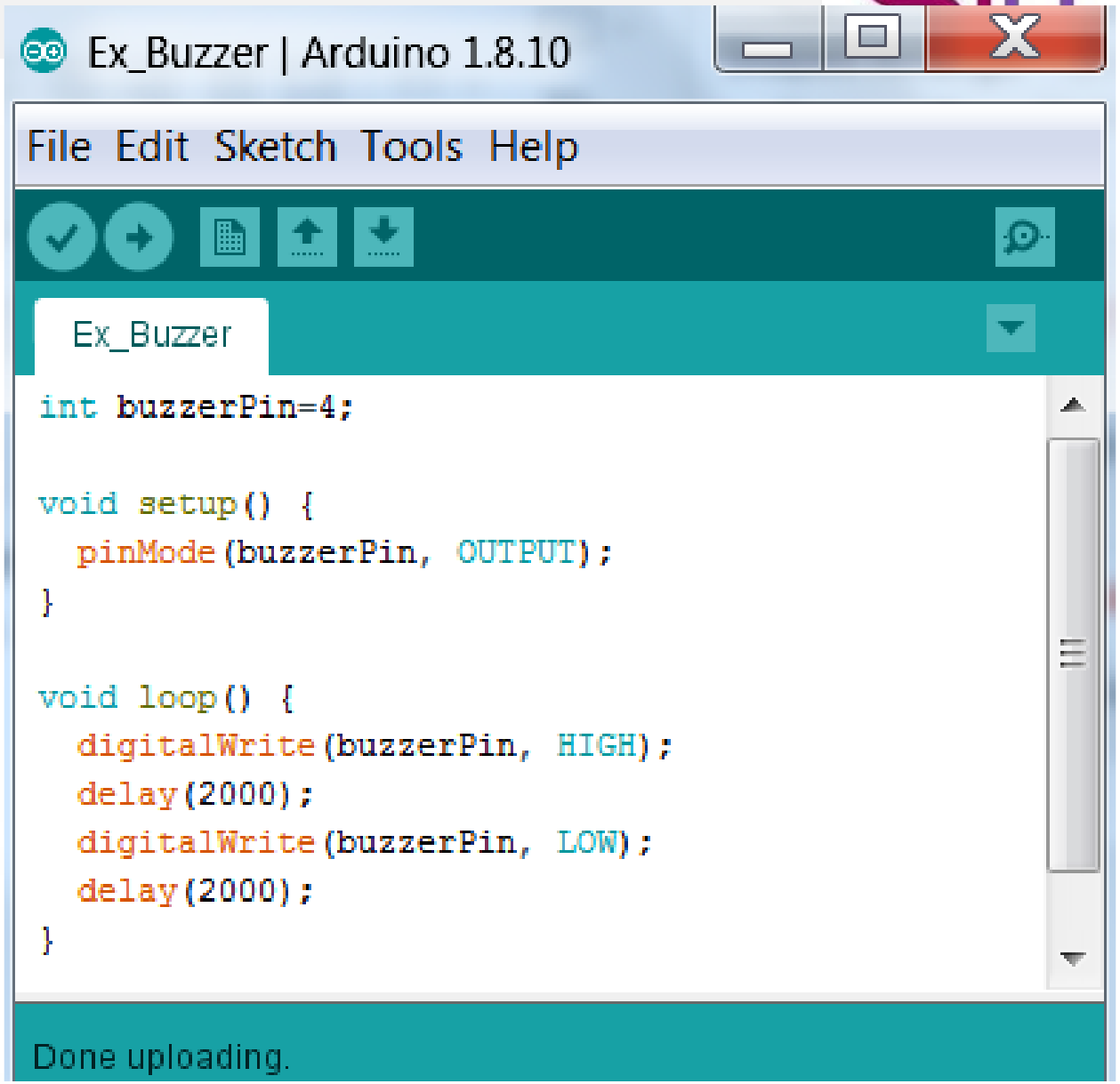
Pin Connection

Buzzer Pin	Arduino Pin
GND	GND
I/O	Digital Pin
VCC	5 V



Buzzer

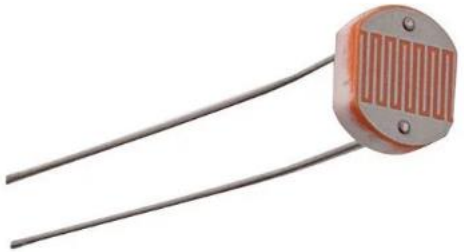
2. An Example of Arduino Sketches: Here, a buzzer is connected to the digital pin 4. The following sketch makes the buzzer to alarm 2 seconds and turns it off for 2 seconds.



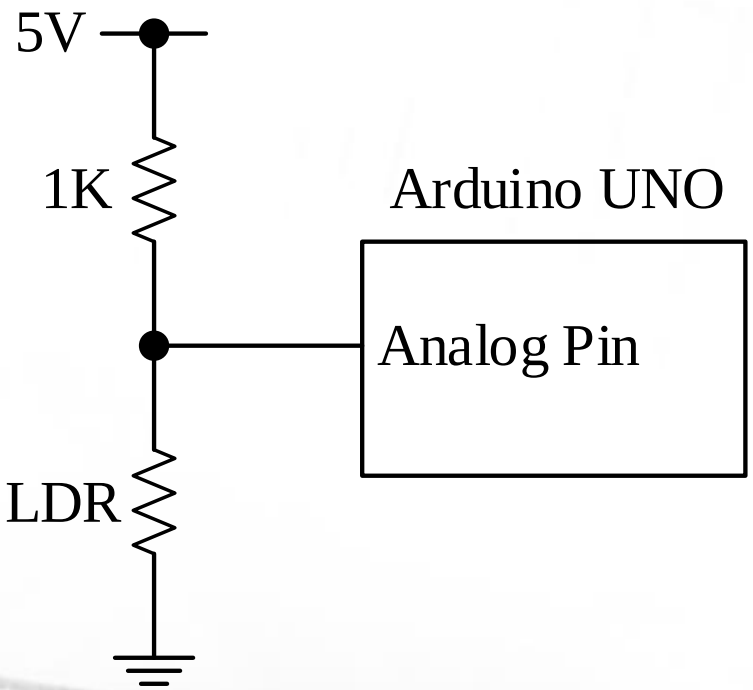
Photoresistor or Light-Dependent Resistor (LDR)

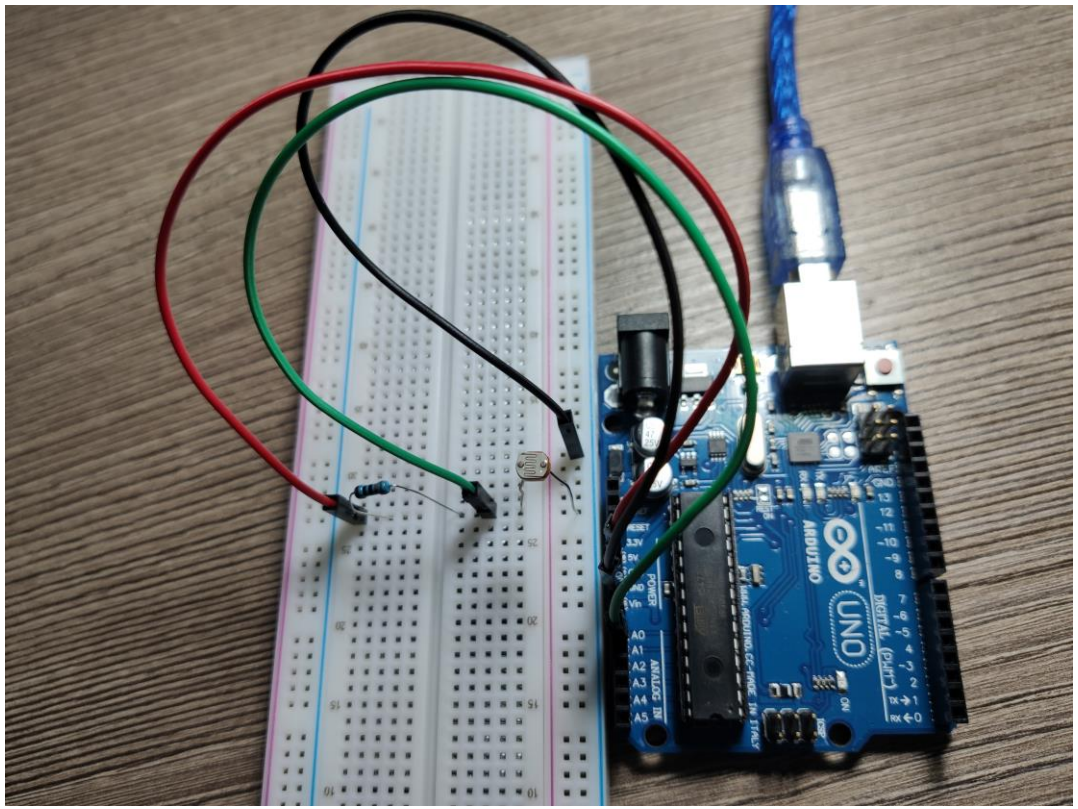
It is a device to measure the intensity (brightness) of light at that area.

- It is a variable resistor:
  - Higher light intensity → Lower resistance



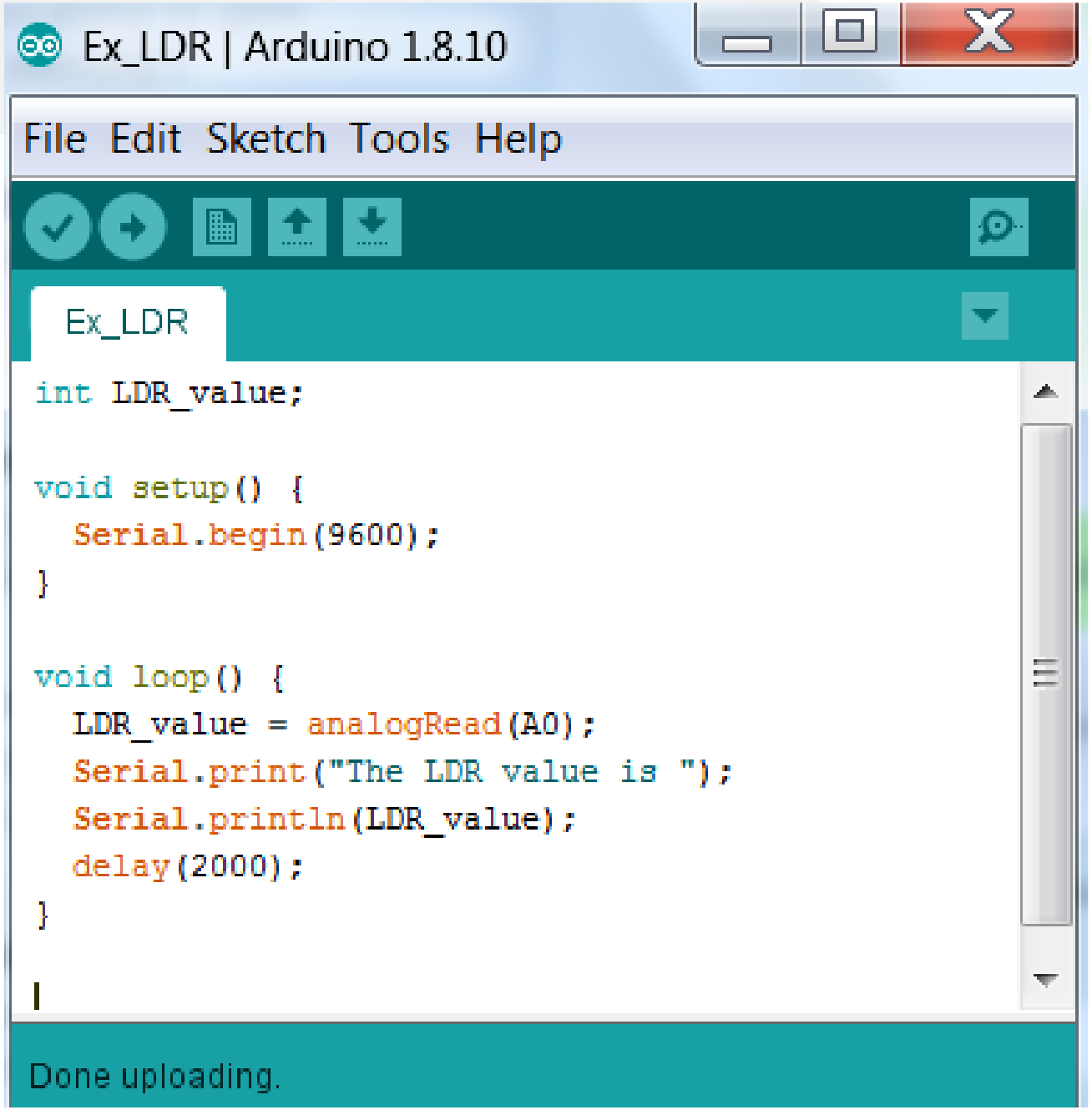
1. Circuit Connection:





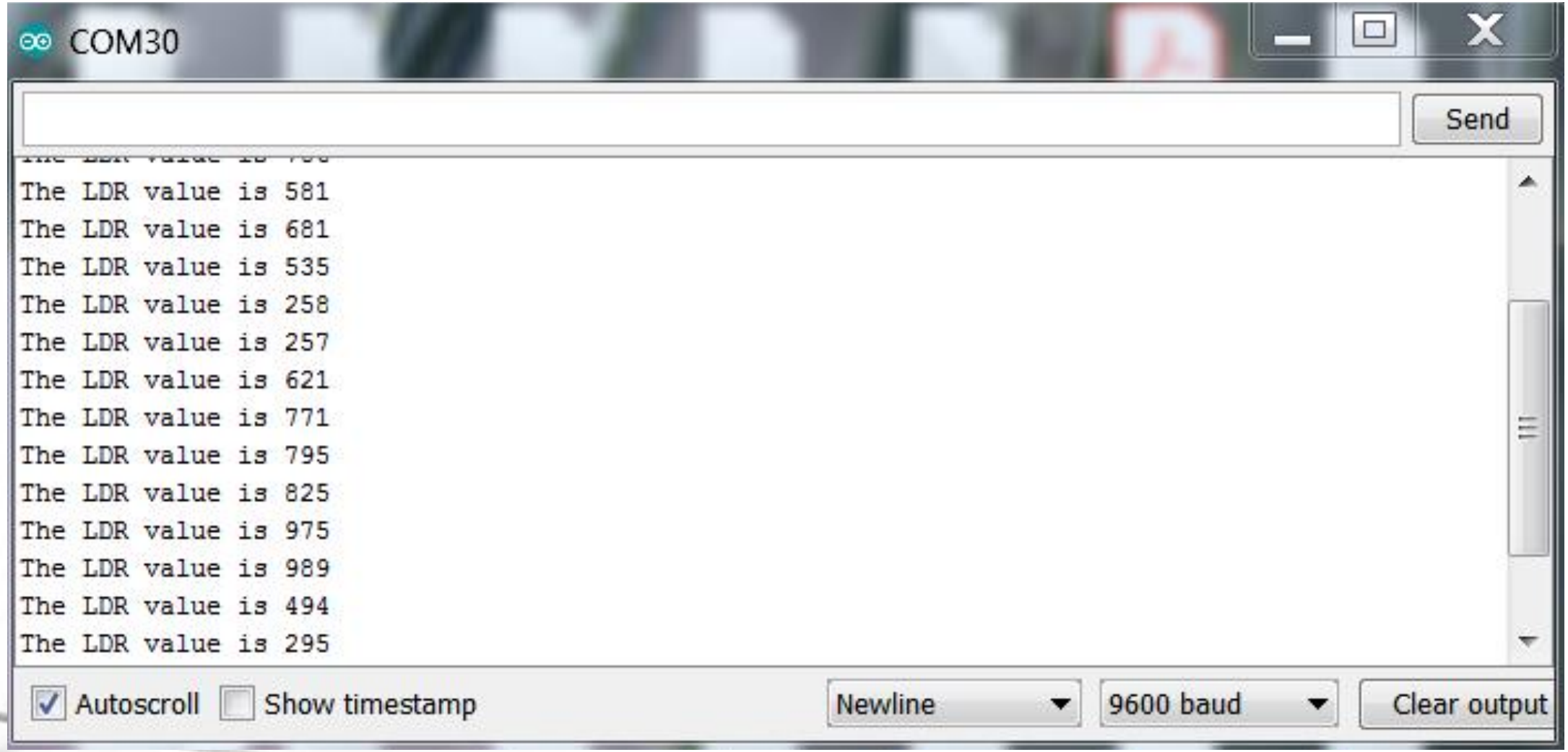
Light Intensity Sensor

2. An Example of Arduino Sketches: We connect the LDR to the analog pin A0. The following sketch reads the input value at the pin A0 and shows on the serial monitor every 2 seconds.



Light Intensity Sensor

As a result, we see the following texts on the serial monitor.



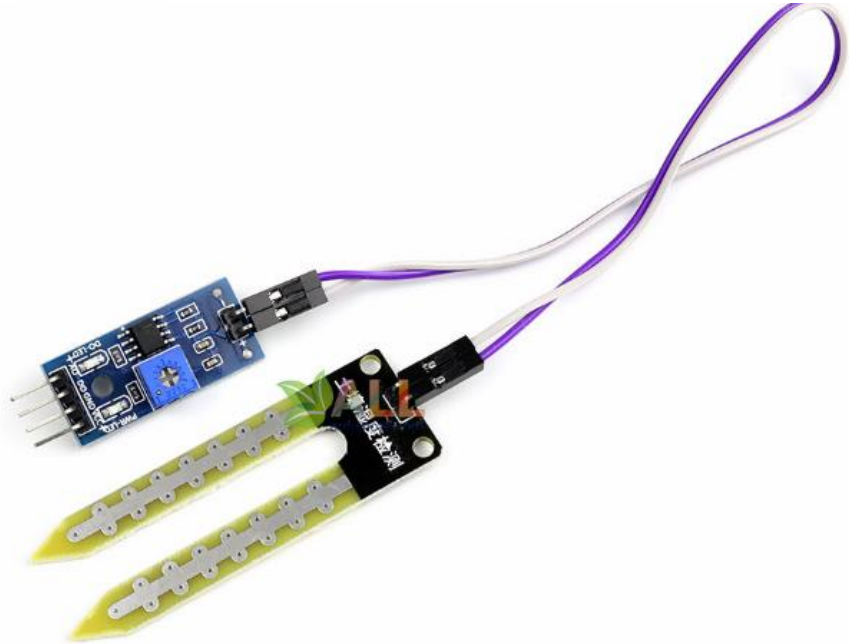
Soil Moisture Sensor

We can use a soil moisture sensor to measure the moisture in soil.

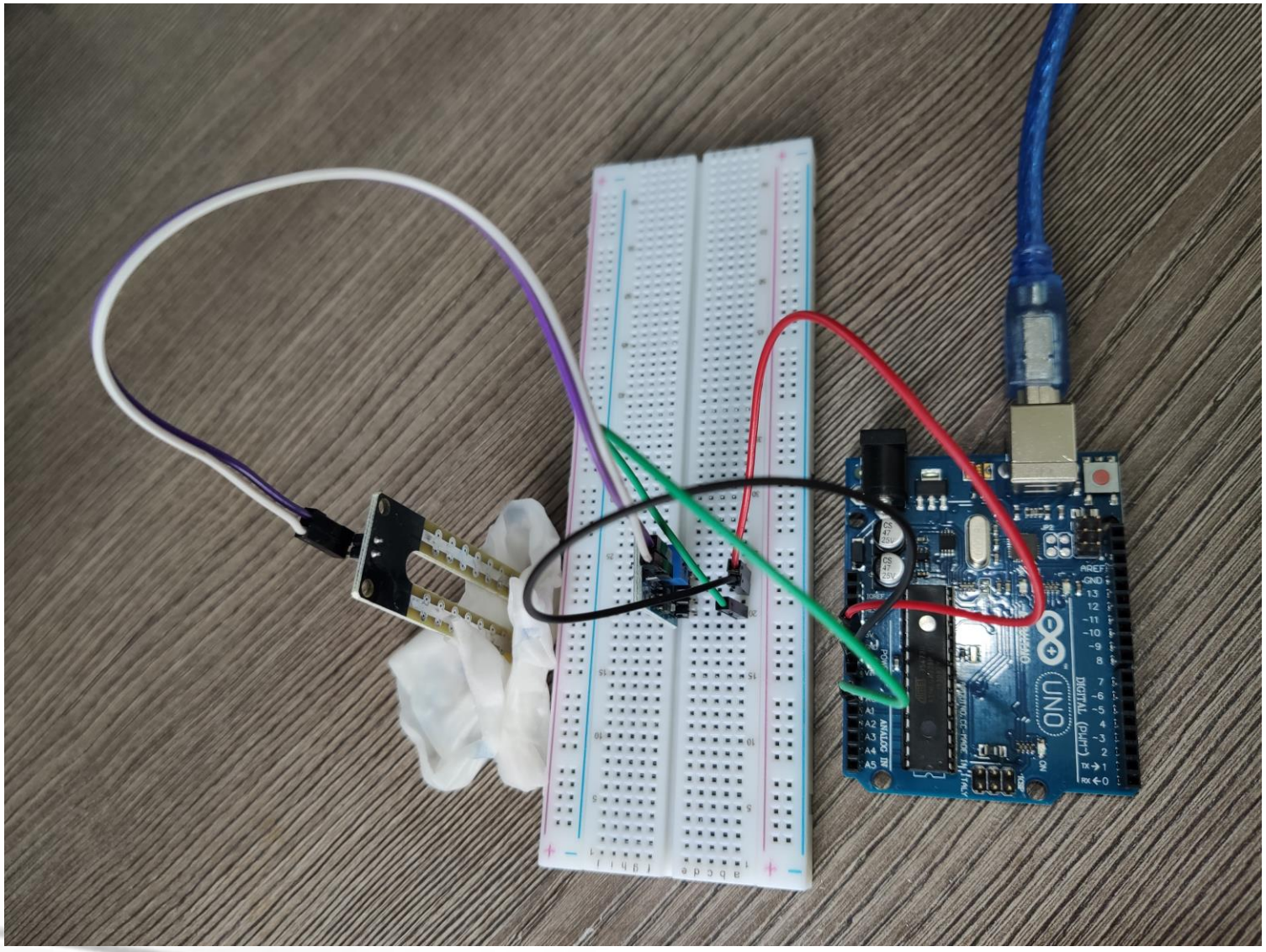
- It works as a variable resistor such that a higher moisture gives a lower value returned by this device.

1. Circuit Connection:

Pin on Soil Moisture Sensor	Arduino Pin
VCC	5V
GND	GND
A0	An analog pin
D0	Not connect

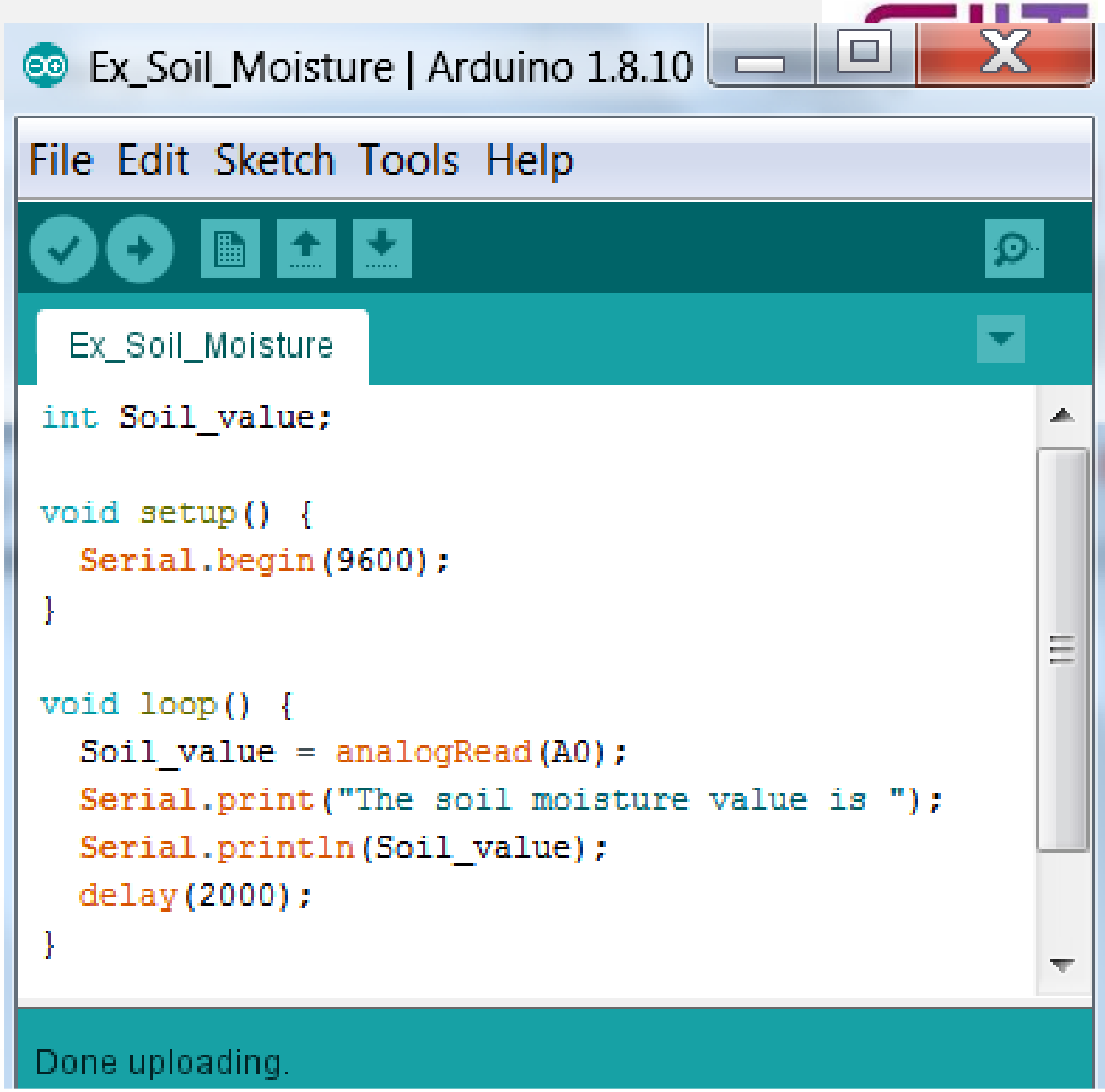


Soil Moisture Sensor



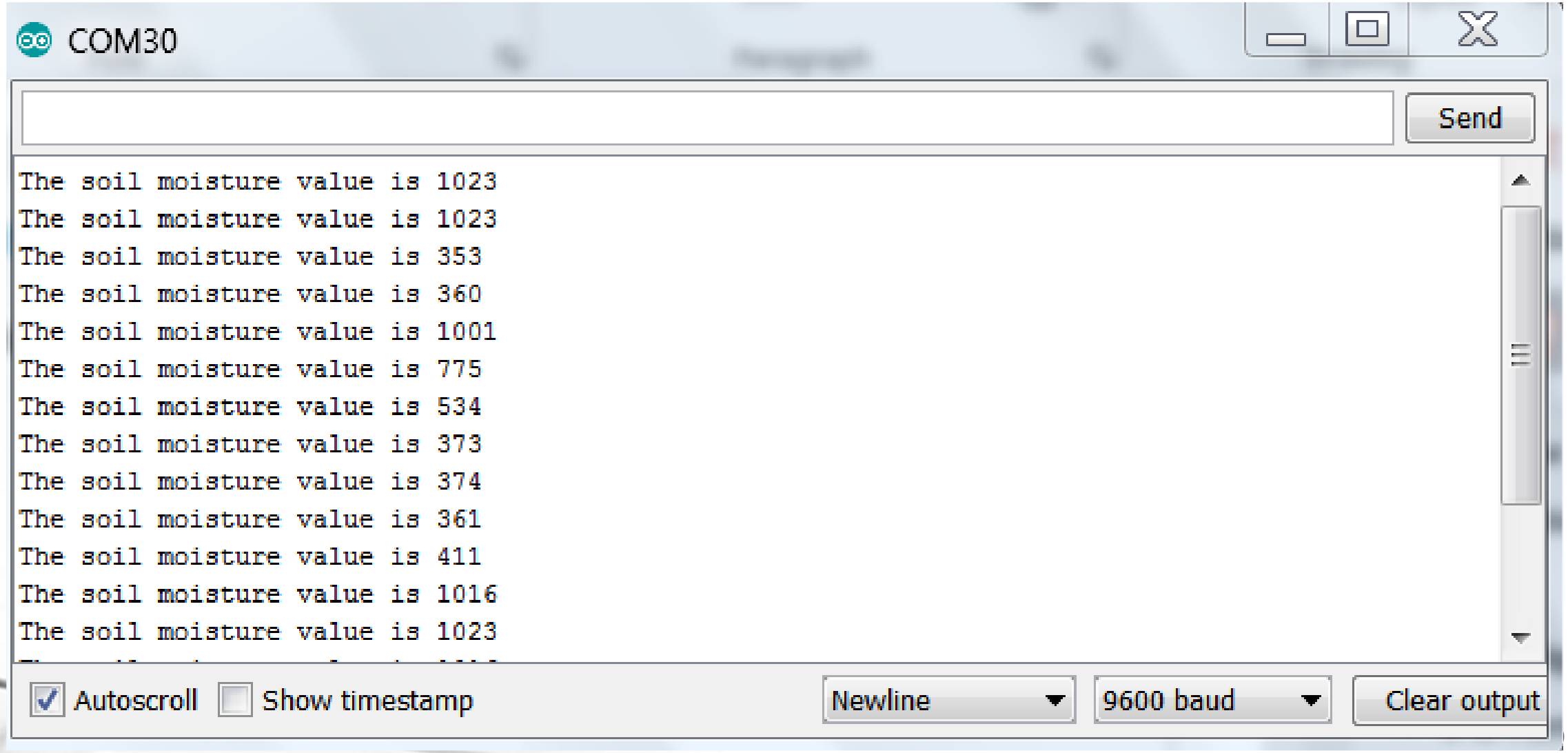
Soil Moisture Sensor

2. An Example of Arduino Sketches: We connect the A0 pin of the soil moisture sensor to the analog pin A0 of the Arduino UNO. The following sketch reads the input value at the pin A0 and shows on the serial monitor every 2 seconds.

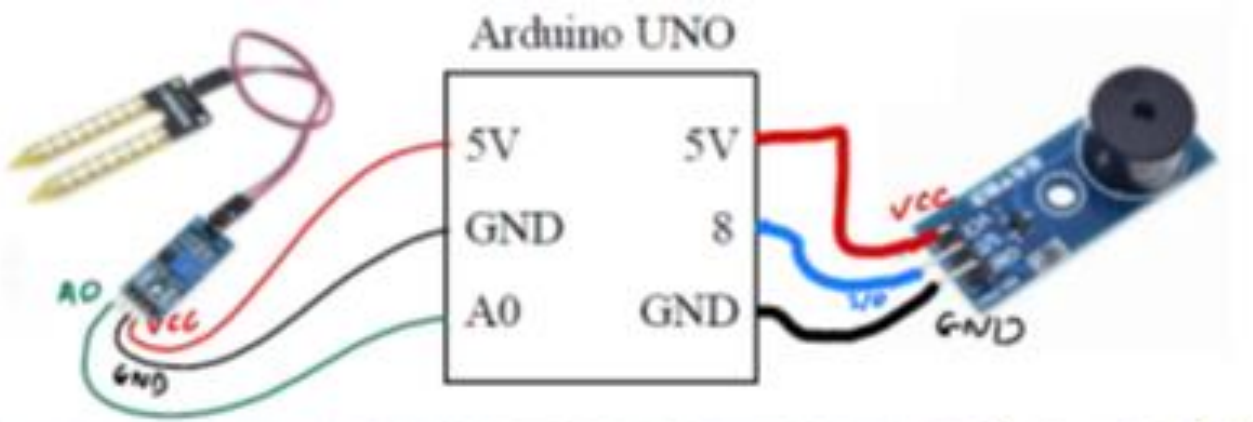


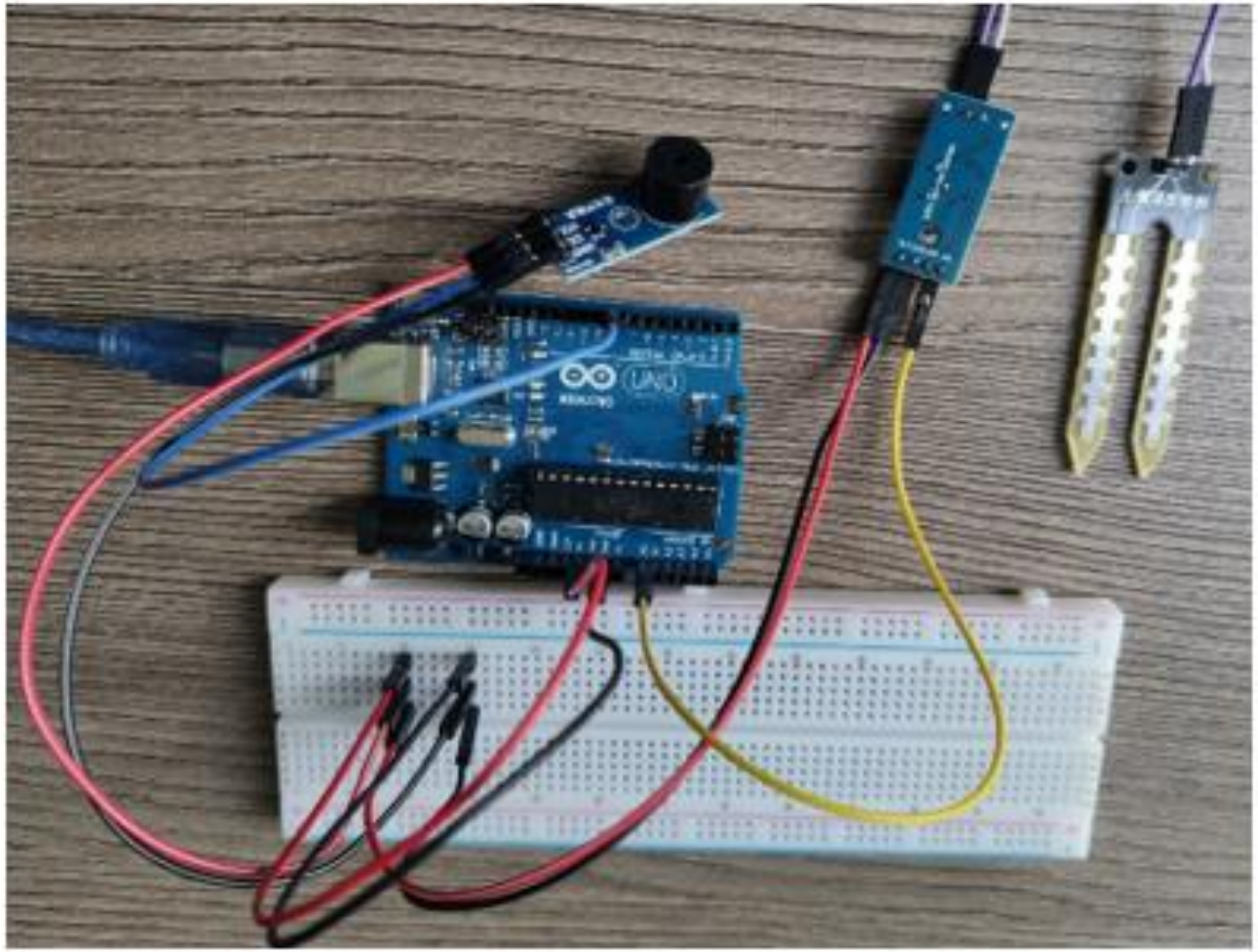
Soil Moisture Sensor

As a result, we see the following texts on the serial monitor.



Ex 4 Soil Moisture sensor & Buzzer





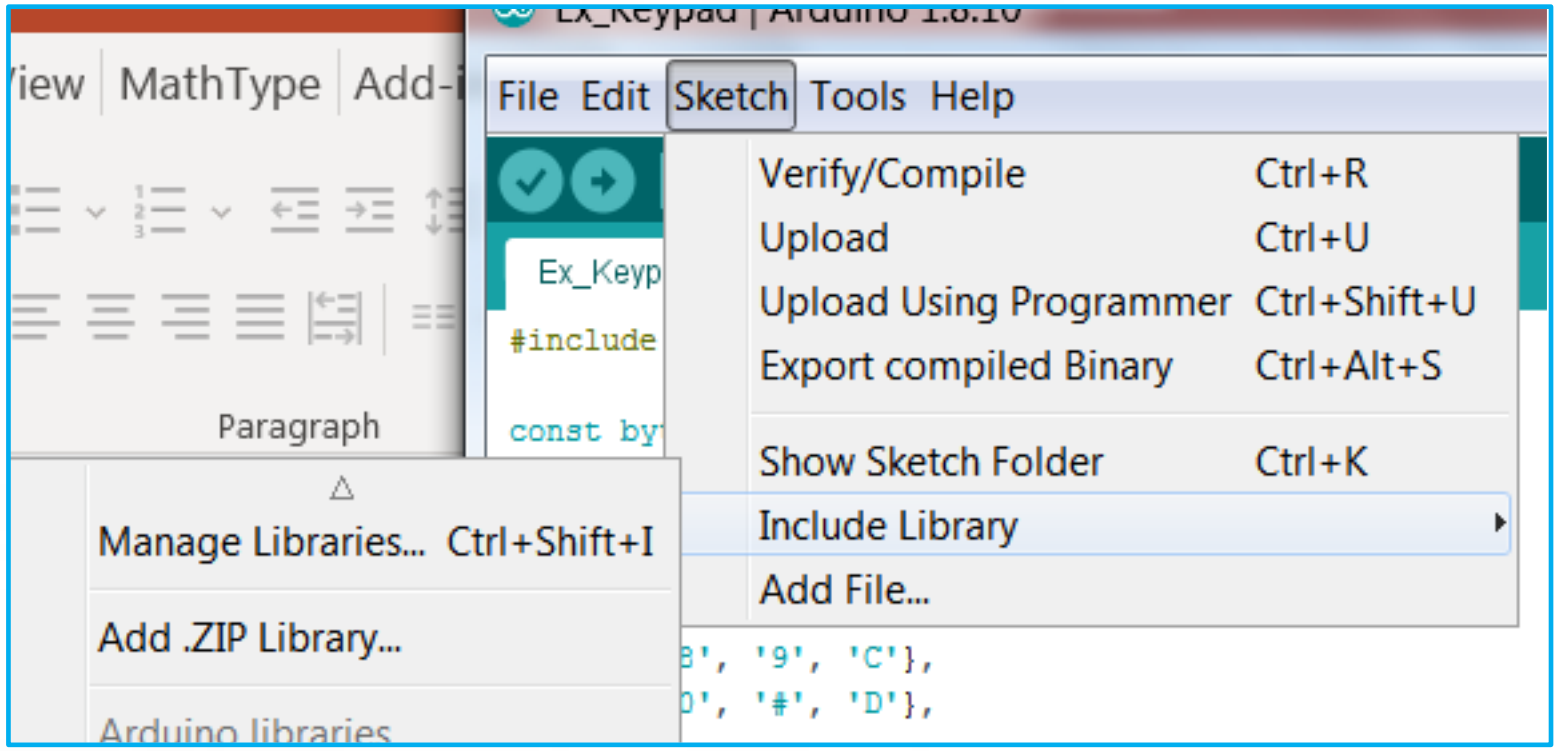
Ex4

```
1  void setup() {
2    Serial.begin(9600);
3    pinMode(8, OUTPUT);
4    digitalWrite(8, HIGH);
5  }
6
7  void loop() {
8    int inValue = analogRead(A0);
9    Serial.println(inValue);
10   if (inValue > 750) {
11     digitalWrite(8, HIGH);
12   } else {
13     digitalWrite(8, LOW);
14   }
15   delay(1000);
16 }
```

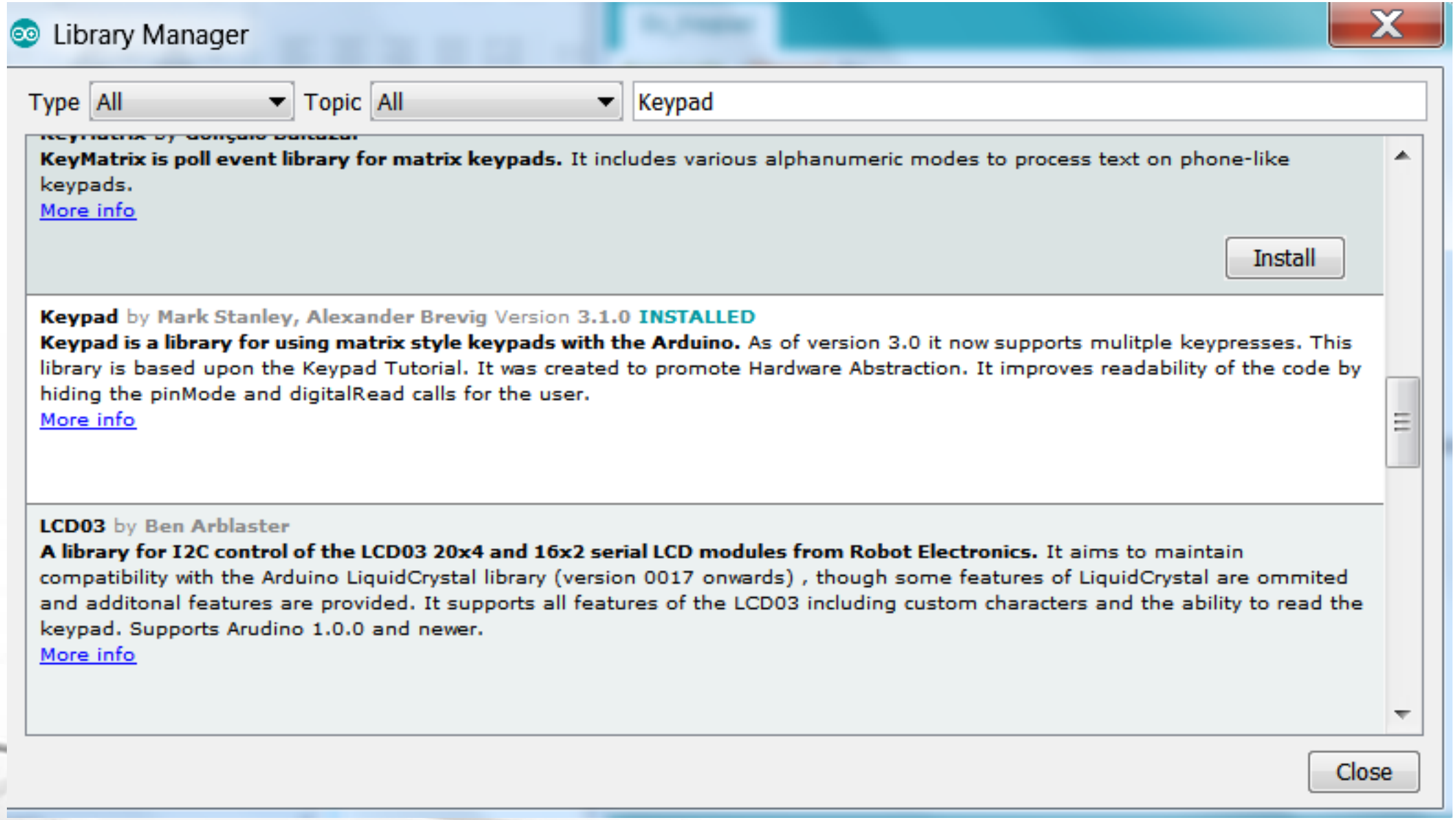
Keypad

To use a keypad we need to install a header file named: Keypad.h

- To install the header file Keypad.h, on the Arduino IDE,
  - Select: Sketch > Include Library > Manage Libraries…
  - Search for: Keypad

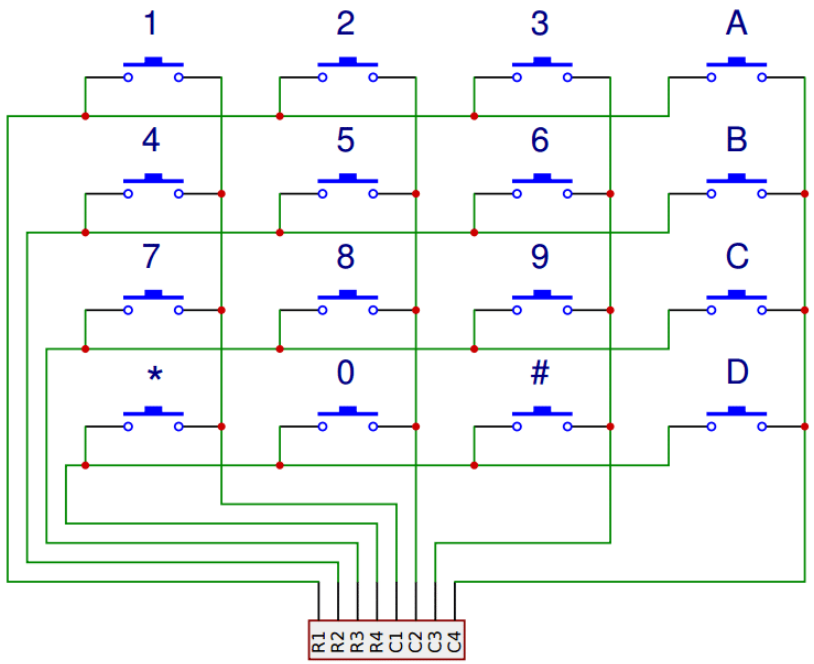


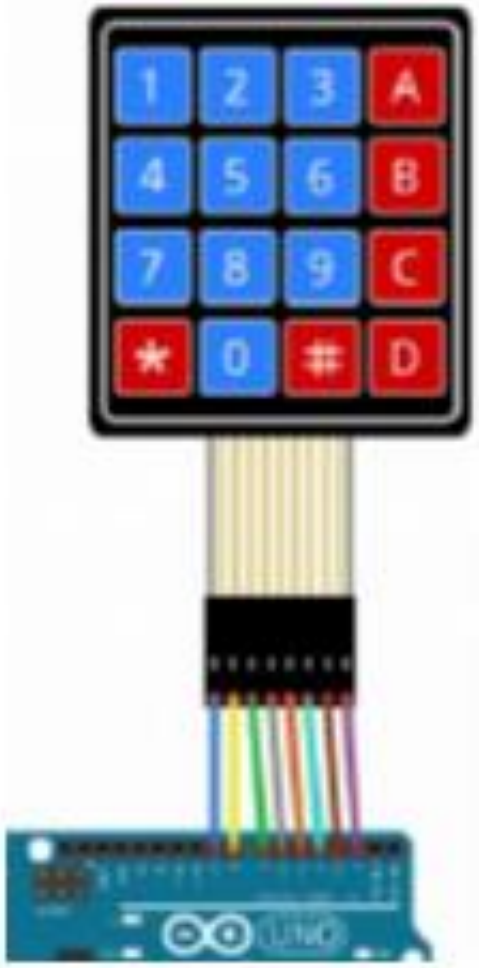
Keypad


Library Manager
Type All
Topic All
Keypad
KeyMatrix is poll event library for matrix keypads. It includes various alphanumeric modes to process text on phone-like keypads.
More info
Install
Keypad by Mark Stanley, Alexander Brevig Version 3.1.0 INSTALLED
Keypad is a library for using matrix style keypads with the Arduino. As of version 3.0 it now supports mulitple keypresses. This library is based upon the Keypad Tutorial. It was created to promote Hardware Abstraction. It improves readability of the code by hiding the pinMode and digitalRead calls for the user.
More info
LCD03 by Ben Arblaster
A library for I2C control of the LCD03 20x4 and 16x2 serial LCD modules from Robot Electronics. It aims to maintain compatibility with the Arduino LiquidCrystal library (version 0017 onwards) , though some features of LiquidCrystal are ommited and additonal features are provided. It supports all features of the LCD03 including custom characters and the ability to read the keypad. Supports Arudino 1.0.0 and newer.
More info
Close

Keypad

1. Circuit Connection:





Pin Connection

Keypad Pin	Arduino Pin
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9

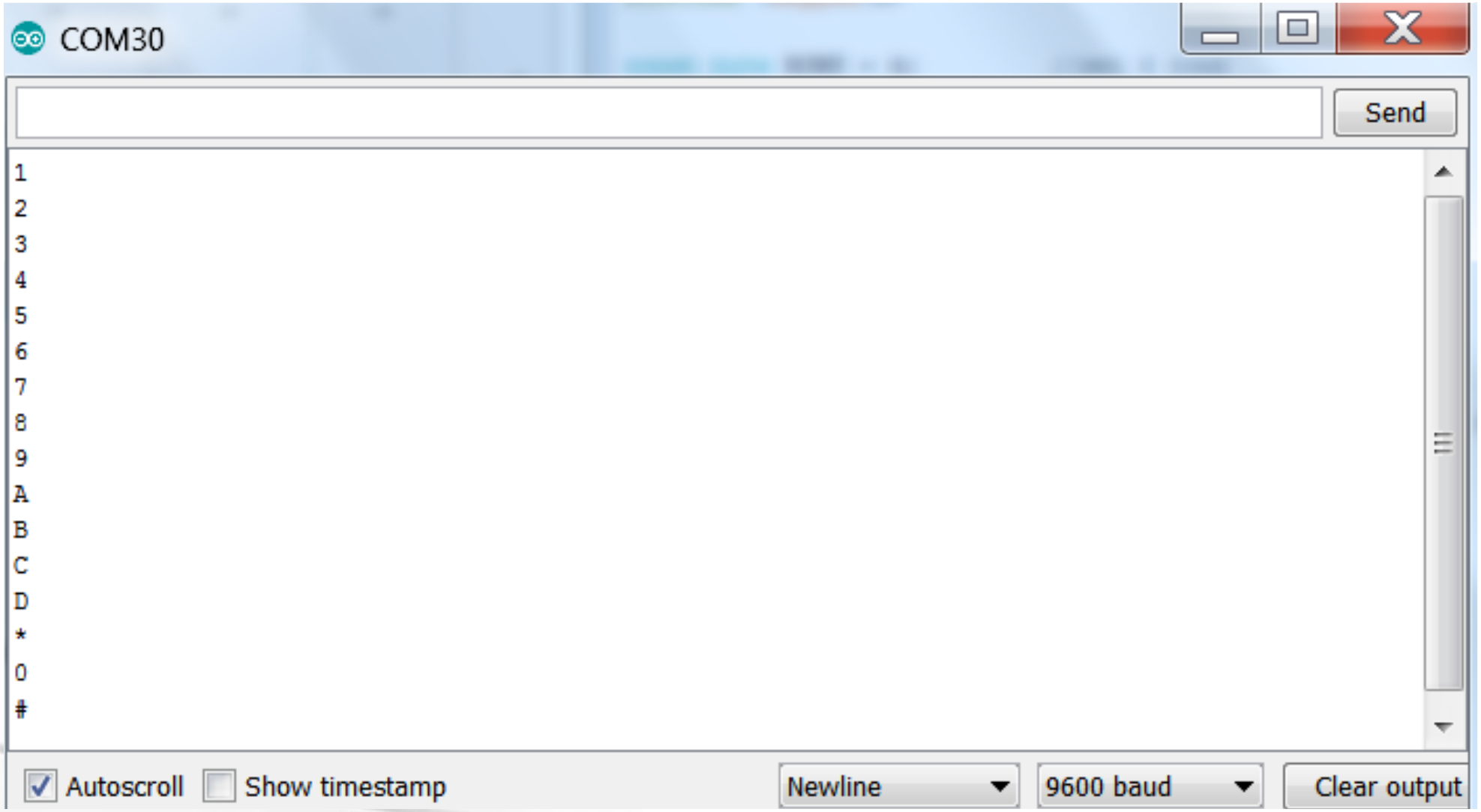
Keypad

2. An Example of Arduino Sketches: We connect the keypad according to the previous page. The sketch in the next page will show the character that we press on the serial monitor.

```
#include <Keypad.h>

const byte ROWS = 4;          //set 4 rows
const byte COLS = 4;          //set 4 columns
char Layout[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'},
};
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};
Keypad customKeypad = Keypad(makeKeymap(Layout), rowPins, colPins, ROWS, COLS);

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

void loop() {
  char customKey = customKeypad.getKey();
  if (customKey) {
    Serial.println(customKey);
  }
}
```

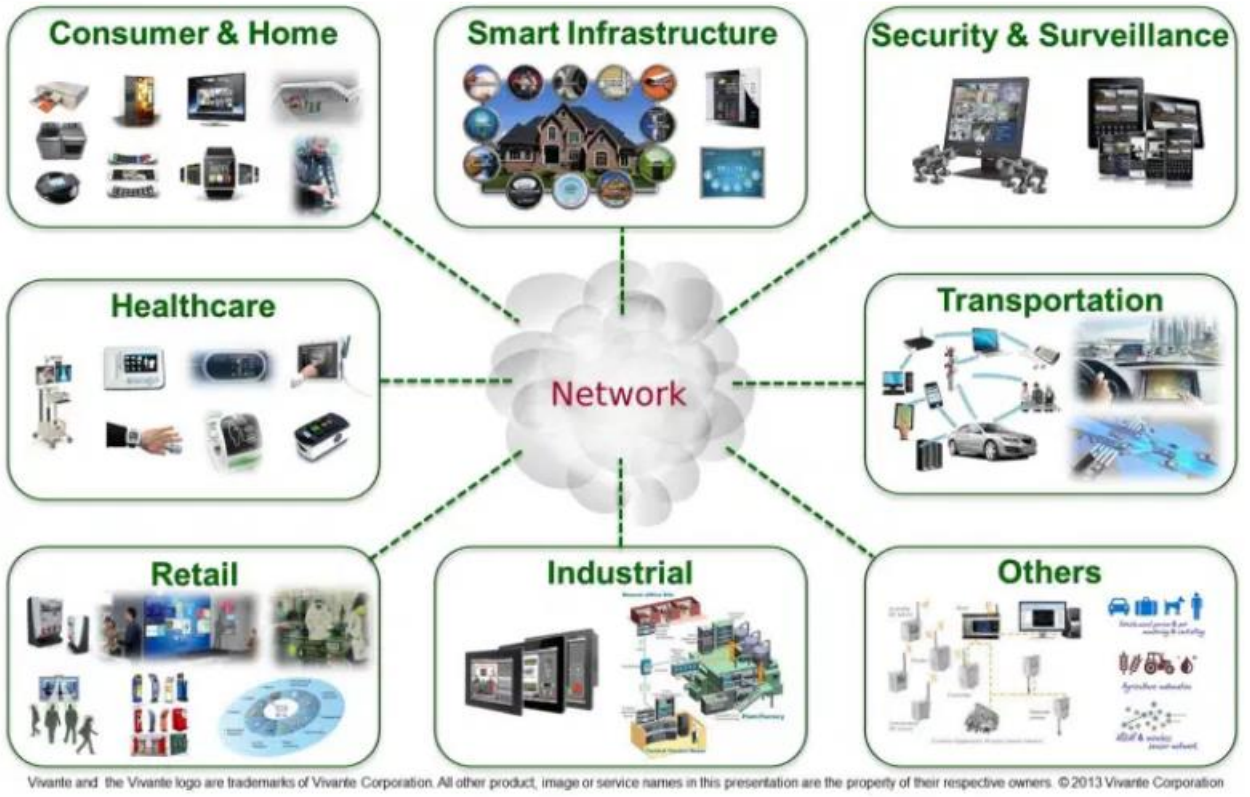
Keypad

As a result, when you press a key on the keypad, that key will be shown on the serial monitor.

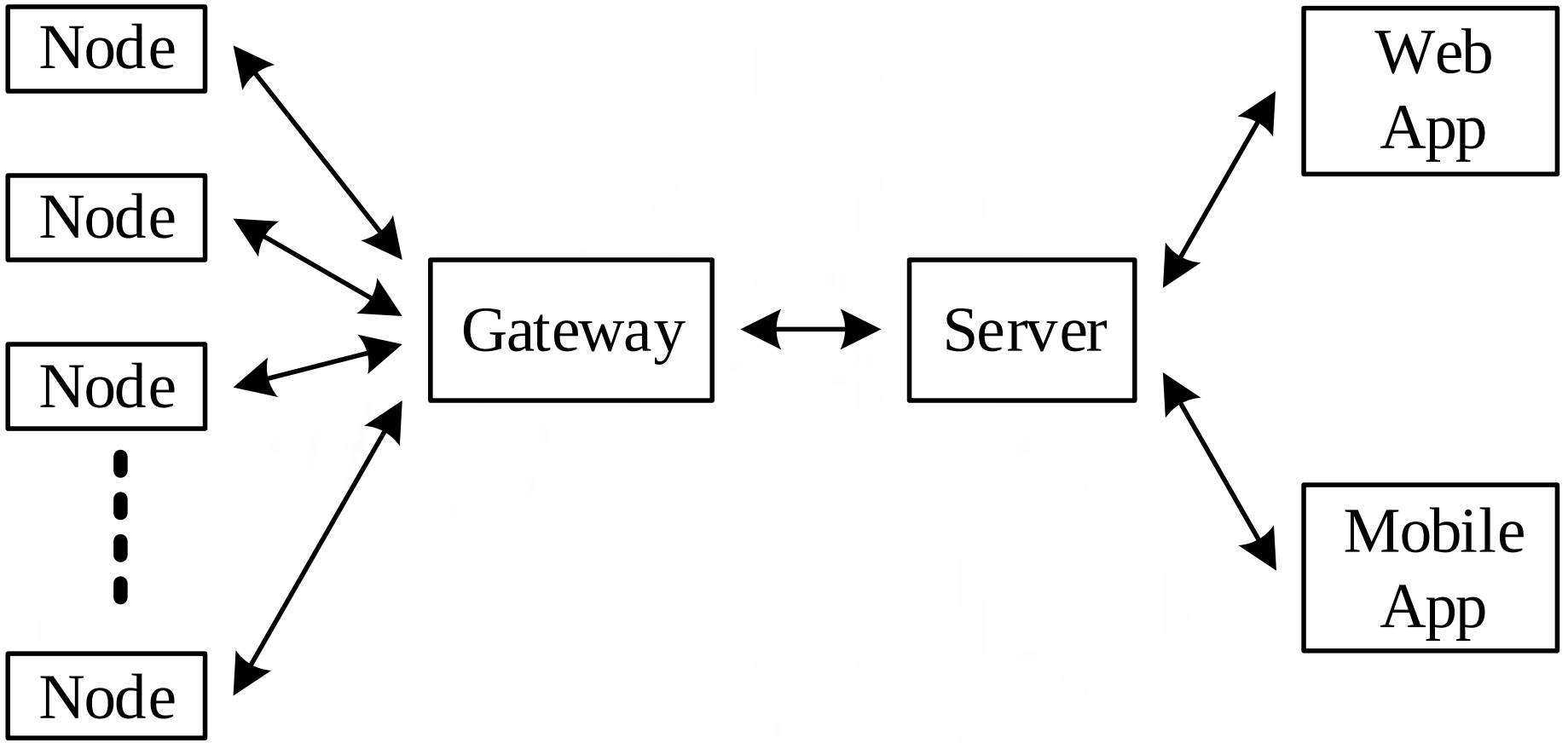


4. Wireless Communication: A Key to the Internet of Things

Internet of Things



Basic Elements of Internet of Things



Wireless Communication Modules

- WiFi
- Bluetooth
- Mobile Communication
- Zigbee
- Etc.

รหัสสินค้า AS10136
ESP-12E (ESP8266) Serial Wifi Transceiver Module

รหัสสินค้า SG10022
HM-10 Bluetooth 4.0 BLE Module CC2541 Chip

รหัสสินค้า SA00040
GSM/GPRS 3G Shield (ElecFreaks)

รหัสสินค้า AS00014
XBee 2mW Wire Antenna - Series 2 (ZigBee Mesh)

Microcontrollers with Communication Ability

รหัสสินค้า BA20001

UNO WiFi Dev Board (ATMega328P + ESP8266 +

รหัสสินค้า BA00021

NodeMCU (Version 2) ESP-12E WIFI Networking

รหัสสินค้า BA00047

ESP32 Development Board WiFi + Bluetooth Dual Core