# Day 01 - Datalogging

## Required parts list

## Introduction to Arduino

### What is Arduino, where does it come from?

Arduino is an open-source electronics platform that allows users to program microcontrollers. It was developed 2003 at Interaction Design Institute Ivrea(IDII) in Ivrea, Italy by Massimo Banzi. From the outset it has been developed as an open-source language that allows easy programming of low-cost microprocessors. This proved to be remarkably successful, and Arduino manages to establish itself as one of the main languages to program microprocessors and for easy prototyping.

A blue circuit board with many different components

Description automatically generated

Above the bar “Arduino” in Ivrea where the founders used to hang out. The image on the right shows the first Arduino. And the webpage is here:

<https://www.arduino.cc/>

### What can it do and what Not?!

Arduino enables the creation of prototypes for microprocessors. This allows the development of various devices such as sensors, robots, and smart IoT devices. It's important to note, however, that Arduino is distinct from computers like the Raspberry Pi. While it lacks the robust processing power and an operating system, it compensates by being more energy-efficient, compact, and straightforward.

Microprocessors come with a specific command sets depending on their type and manufacturer. Arduino's commands are constructed "on top" of the existing command set, leading to a couple of implications:

1. Arduino programming is not geared towards professional production. It employs a simplified language designed for prototyping, lacking the comprehensive features of a microprocessor's full command set (though these extra features are often not essential for most applications).
2. Not every microprocessor can be programmed using Arduino; only those with an additional layer of commands specifically written for Arduino compatibility can be utilized.
3. The same script can work on different microprocessors.

### Language

The Arduino project consist of two parts: the language and the hardware. Both are linked together, given that the language can only work on hardware that “knows” Arduino.

The language is based on the “C Language “a low-level language that contains a few simplifications.

The programming environment is called Arduino IDE. This is not only a text editor, but it also translates the code into the specific language of the microprocessor. Consider that each chip and each chip manufacturer have their own commands or methods and it is exactly this translation feature that makes Arduino so accessible and successful.

Practically, you will need to tell the IDE which microprocessor you are working with. For each new microprocessor you like to work with, you will need to ensure that you work with the right configuration in the IDE – it it’s not there you will need to download/install it.

### The board

There is a huge number of different boards. Depending on the project you want to do, you might find a board that is more tailored to your needs. However, the Arduino Uno is an all-purpose entry board that allows you to do a large variety of projects

<https://www.arduino.cc/en/Main/Products>

The components of an Arduino Uno Board

<https://www.hackerearth.com/blog/developers/a-tour-of-the-arduino-uno-board/>

The full pin out diagram:

<https://www.circuito.io/blog/arduino-uno-pinout/>

…and the circuit on the board (for reference):

https://www.arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf

Arduino is an open-source project. This means that everything is open and transparent. Everybody can use it and there is no restriction in how you can use Arduino. This has led to many other companies that produce Arduino compatible boards. Just a few, in no order:

<https://www.adafruit.com/>

<https://www.sparkfun.com/>

https://www.seeedstudio.com/

### The IDE & First Sketch

* Open first sketch from examples
* Select your board type and port
* Upload the program

## Introduction to Coding in Arduino (C Language)

### 000 – Anatomy

* Libraries
* Declare variables
* Void setup()
* Void loop()
* Functions

I will run you though a few code examples to show you the most important features of the Arduino C language.

It is always a good idea to check the reference page:

<https://www.arduino.cc/reference/en/>

Arduino compiles the code in the IDE like this:

Diagram

Description automatically generated

### 001 - Basic LED Blink Sketch

Anatomy of a sketch, basic intro, commands, setup, and loop

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_BUILTIN, HIGH);

delay(1000);

digitalWrite(LED\_BUILTIN, LOW);

delay(1000);

}

### 002 - Basic LED Blink Sketch with Variable and Feedback over the Serial Monitor

Introduction of a variable and output over the serial monitor. Given that we are going to declare our first variable here, it is worth looking at the typical datatypes in C.

1 byte is 8 bits (00000001)

1 is 00000001

2 is 00000010

…

256 is 11111111 (the largest possible value in byte)

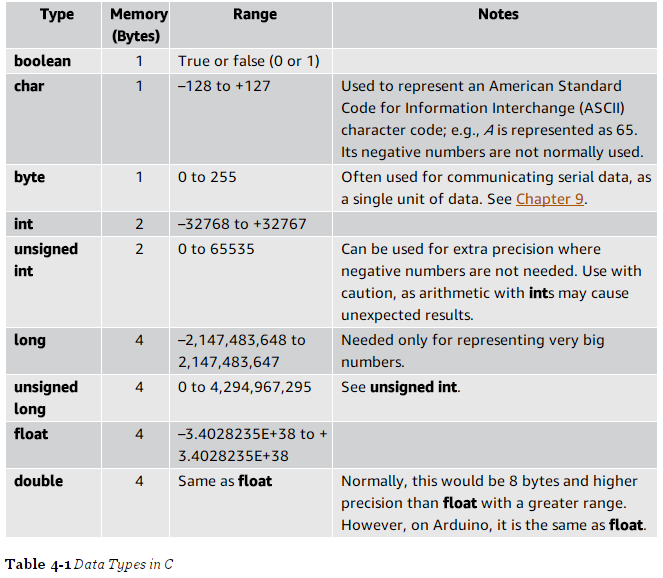
The Arduino Rev 3 (and also other types) has different types of memory:

EEPROM 1kb Permanent memory

RAM 2KB temporary data or run-time date such as variables are stored

FLASH 32KB this is where the code is saved

And this is the of different datatypes. The typical datatypes are **boolean**, **char**, **int** and **float**.



**int pause = 50;**

void setup() {

Serial.begin(9600);

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

digitalWrite(LED\_BUILTIN, HIGH);

delay(pause);

digitalWrite(LED\_BUILTIN, LOW);

delay(pause);

Serial.print("The delay is: ");

Serial.print(pause);

Serial.println("ms");

}

### 003 - Blink Sketch with a Function

Introduction of a simple function to keep the code organised, shorter and modular

void setup() {

Serial.begin(9600);

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

**flash**(50, LED\_BUILTIN);

**flash**(500, LED\_BUILTIN);

}

**void flash (int pause, int ledNumber) {**

**digitalWrite(ledNumber, HIGH);**

**delay(pause);**

**digitalWrite(ledNumber, LOW);**

**delay(pause);**

**Serial.print("The delay is: ");**

**Serial.print(pause);**

**Serial.println("ms");**

**}**

### 004 - Control Structure with “if”

If control structure with a simple conditional statement

bool Fast = false;

void setup() {

Serial.begin(9600);

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

**if (true == Fast) {**

**flash (50, LED\_BUILTIN);**

**}**

**else {**

**flash (400, LED\_BUILTIN);**

**}**

}

void flash (int period, int led) {

digitalWrite(led, HIGH);

delay(period);

digitalWrite(led, LOW);

delay(period);

Serial.print("The delay is: ");

Serial.println(period);

}

### 005 - Iterations with the “for” – loop

For loop:

void setup() {

Serial.begin(9600);

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

**for (int i = 0; i < 10; i = i + 1) {**

**Serial.print("Loop Nr.");**

**Serial.print(i);**

**Serial.print("\t");**

**flash (20, LED\_BUILTIN);**

**}**

}

void flash (int period, int led) {

digitalWrite(led, HIGH);

delay(period);

digitalWrite(led, LOW);

delay(period);

Serial.print("The delay is: ");

Serial.println(period);

}

### 006 - For loop with array

Array syntax and how to call a number in an array

**int timedelay[] = {200, 40, 50, 500, 70};**

void setup() {

Serial.begin(9600);

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

for (int i = 0; i < 5; i = i + 1) {

Serial.print("Loop Nr.");

Serial.print(i);

Serial.print("\t");

Serial.print("Number from array: ");

Serial.print(timedelay[i]);

Serial.print("\t");

flash (timedelay[i], LED\_BUILTIN);

}

}

void flash (int period, int led) {

digitalWrite(led, HIGH);

delay(period);

digitalWrite(led, LOW);

delay(period);

Serial.print("The delay is: ");

Serial.println(period);

}

### 007 - Communication via the serial monitor

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

digitalWrite(LED\_BUILTIN, LOW);

Serial.begin(9600);

}

void loop() {

if (Serial.available() > 0) {

char letter = Serial.read();

if (letter == '1') {

digitalWrite(LED\_BUILTIN, HIGH);

Serial.println("LED is on!");

}

else if (letter == '0') {

digitalWrite(LED\_BUILTIN, LOW);

Serial.println("LED is OFF!");

}

}

}

### 008 - Debugging

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

digitalWrite(LED\_BUILTIN, LOW);

Serial.begin(9600);

}

void loop() {

if (Serial.available() > 0) {

char letter = Serial.read();

if (letter == '1') {

digitalWrite(LED\_BUILTIN, HIGH);

Serial.println("LED is on!");

}

else if (letter == '0') {

digitalWrite(LED\_BUILTIN, LOW);

Serial.println("LED is OFF!");

}

if (digitalRead(LED\_BUILTIN) == LOW) {

Serial.println("The LED is really OFF");

}

else if (digitalRead(LED\_BUILTIN) == HIGH) {

Serial.println("The LED is really ON");

}

}

}

### 009 - Messages in the serial Interface

void setup() {

Serial.begin(9600);

}

void loop() {

String message = "";

if (Serial.available() > 0) {

while (Serial.available() > 0) {

message = message + char(Serial.read());

delay(250);

}

Serial.println(message);

}

}

## Communication with external devices

Your Arduino has several ways of communicating with external devices. Each of these ways are different and it depends on the circumstances which communication protocol you are choosing. Usually, it is not your choice when to use what – it depends on the items you want to connect.

We already used *serial communication* in the sketches above, and we will have a look at *I2C* and *SPI*.

Here an overview about the standard assignment of pins:

https://maker.pro/arduino/tutorial/common-communication-peripherals-on-the-arduino-uart-i2c-and-spi

### Serial Communication

Serial Communication is also called *UART (Universal Asynchronous Reception and Transmission)*. This is the most basic one. It is also the one that communicates with the computer over USB and goes both ways. There are two pins assigned for this protocol - they are hardwired and called TX/RX. You see that they are assigned to the ports 0 and 1. There are also two LED’s on the boards that blink when there is activity over those ports.

Because it’s hardwired, it is called *hardware serial*. The main commands have been shown in the sketches above, but there are a few more:

https://www.arduino.cc/reference/en/language/functions/communication/serial/

The Arduino Uno has one hardware serial interface, whereas the Arduino Giga has four.

If you use hardware serial to communicate to your computer (and that happens rather often) then you cannot use these ports for anything else. You cannot “double use” pins and you cannot communicate to a sensor and communicate to your computer over the same pins (even if it’s not at the same time).

This leaves us with two alternatives:

1. Buy a new board, some boards have many hardware serial interfaces, the Arduino Mega has three.
2. But with the use of a library called “software serial” you can emulate a serial communication on any pins you like. This does not conflict with the communication to the computer:

<https://www.arduino.cc/en/Reference/softwareSerial>

Serial communication has two pins, RX and TX that are being cross connected to the sensor. i.e. RX > TX and TX > RX.

### I2C

We will not be using this protocol for our purposes and thus I will keep it rather short. Yet there are a few sensors that use it and you might end up with one.

I2C is a protocol that allows you connect multiple sensors to the same set of pins – theoretically 128 sensors to three pins and the same set of wires. This is promising, yet this protocol is not particularly fast.

The I2C protocol involves using two lines to send and receive data: a serial clock pin (SCL) that the Arduino Master board pulses at a regular interval, and a serial data pin (SDA) over which data is sent between the two devices. The third line is the power.

The protocol works with a Master – Slave Hierarchy. Each slave has a unique address that allows the master to control the sensor.

A library called “wire” allows us to include the protocol into this sketch.

https://www.arduino.cc/en/reference/wire

### SPI

Serial Peripheral Interface (SPI) is the default protocol for the communication with SD cards or display modules in the Arduino world. The advantage is that it is fast, but there is only a limited number of four devices. There are also more cables.

<https://www.arduino.cc/en/Reference/SPI>

SPI requires the definition of four pins, three of them are common to all devices and can be shared, the fourth pin is specific to the device. On the Arduino Uno Board digital pins 11, 12, and 13 are used for this protocol. These pins are doubled up: they exist as regular pins, but they also exist on the ICSP (the six pins on the right-hand side of the board). This protocol is usually straightforward, we have to do two things to use this protocol: reference to the SPI library and select the correct pin SS (Slave Select), CS (Chip Select) or similar.

## Groove Sensors

### Intro

Grove sensors is a line of sensors of a manufacturer called Seeedstudio.

<https://wiki.seeedstudio.com/Grove_System/>

This system uses standard connectors that make it easy to connect sensors or motors to the Arduino. A simple and practical system for fast prototyping.

### How Do They Work?

The cables go into a special slot on the shield that you have bought. These slots are connected to the electricity and to a certain pin that is indicated on the board. All we have to do is to make sure that the code refers to the correct pin.

## Groove Air Quality Sensor

### Intro

The first sensor we are going to look at is the groove air quality sensor.

https://wiki.seeedstudio.com/Grove-Air\_Quality\_Sensor\_v1.3/

### 010 Code Air

Example Code

## GROVE GSR SENSOR

### Intro

Galvanic Skin Response

https://wiki.seeedstudio.com/Grove-GSR\_Sensor/

### 011 Code GSR

Example Code

## Grove Dust Sensor

### INTRO

The dust sensor of grove is based on the low-cost dust sensing unit Shinyei PPD42NS. It aspires the air through a heating unit and measures its opacity over a given timeframe with a photosensitive resistor arrangement. For the unit to work property it must heat up for about 3 minutes and be upright. Bear in mind that electronic heating elements take quite a bit of current, please factor that in when your sensor is battery based

The unit recognises particles with a diameter of more than 1μm. A hair is about 70 μm, pollen about 10 μm and smoke particles are about 2.5μm. The value we get is basically the concentration of dust particles in the air, but not more (We will not understand what kind or size we have).

The samples every 30 seconds and it returns a value of particles per 1/100 of a cubic foot (pcs/0.01cf).

The values range from 0-8000 pcs/0.01cf and on the [internet](https://funprojects.blog/2017/05/31/dust-monitor/), I found following scale:

0-500 pcs/0.01cf = a clear room

500-1500 pcs/0.01cf = a “fairly” clean room

1500-4000 pcs/0.01cf = a room in need of dusting (but not super dusty)

4000+ pcs/0.01cf = if you have allergies, you may leave the room

Values at my place were about 700, that seems to validate the ranges above.

### 012 Code Dust

The unit connects to one of the digital slots on the grove board. In this case we put it to slot D8 that connect to the pin 8.

[example code “Dust Sensor”]

## GPS Tracking on Arduino

### Into

GPS is the shortcut for “Global Positioning System”, this allows a GPS receiver to get geolocation and time information. This relies on an unobstructed view from the GPS receiver to at [least four satellites](https://timeandnavigation.si.edu/multimedia-asset/how-does-gps-work). The precision geolocation depends on the quality of the signal. The quality of the signal depends on how good the connection to the satellites is. The receiver itself is passive, he just receives data, but doesn’t transmit anything back.

Many GPS receives data using a so called [NMEA 0183](https://en.wikipedia.org/wiki/NMEA_0183) protocol and this is also the case for our receiver. It is not possible to just “read” in a text editor or similar, there are steps in-between. To successful “read” or parse the signal, we need to install an external library that allow us to print out the values in text form.

As a minimum, you can get the time and your position from a typical gps signal. But there are also other data such as speed, altitude and the heading among other data.

There are many different models on the market, but I am going to focus onto the one that you have.

* Regular GPS Module on break out board
* Grove GPS Module

### Datasheet

Generally, the GPS modules uses serial communication. This means that it is a data protocol that relies on two pins: TX and RX. Tx is transmit and Rx is receive, - note that TX/RX need to be cross connected. RX connects to TX and TX connects to RX. There are two other pins, VCC and GND, and this is for the power supply. S

Almost all Arduino gps modules have these four pins and whist the wiring might look different, the code does not so much. The chip itself looks straightforward, there is an antenna and a module, in our case it is a NEO-6-SERIES module. These modules are being produced by uBlox and it is worth the time to investigate the website of the manufacturer:

https://www.u-blox.com/en/product/neo-6-series

After we have checked the website of the chip manufacturer, it’s worth to familiarise yourself with the website of the one that has sold you the module:

<https://www.waveshare.com/uart-gps-neo-7m-c-b.htm>

<https://wiki.seeedstudio.com/Grove-GPS/>

Here we will be looking for more information, like pin assignment, voltage, schematics etc.

Let’s connect the GPS receiver to the board.

It might appear easier to plug in the TX into the RX pin of the Arduino board, but this is a bad idea. Once connected, the gps module starts for feed a stream of data. Given the connection properties of the RX and TX pins on the Arduino, this would cause problems and it’s much better to connect the module to a set a generic digital pin. We will need to tell those pins that they are going to work as serial communication pins. We do that with a library that is called serial library. It basically replicates the hardware RX/TX, but as extra software RX/TX – this allows us to control the stream a bit better and we can have multiple RX/TX at the same time.

### GPS Library

In addition, we need to install a library that allows us to parse the incoming signal. In our case, the “TinyGPS++” library can be downloaded as zip and installed. There are many others, but this one work well for our purpose

<http://arduiniana.org/libraries/tinygpsplus/>

In a nutshell, the code runs though these steps:

* Establish a new serial connection to gps module and give it a name.
* Create a gps object.
* Check if this object is valid and if it has been updated

This this is all the case, query this object for the specific info, like date, time, position, etc

Output the information to the other software serial connection that goes to your computer

### 013 Code GPS Module

Code Example

### 014 Code GPS STRING FORMAT

Code Example

The code has some characters that might look unfamiliar to you: “\n” or “\t”. They are called escape characters and they are used to perform basic formatting. They will not be printed, and the computer reads these characters in a specific way. “\n” is a new line and ”\t” is a tap. It is a good idea to keep these in mind.

## Writing Data on SD Card

### Intro Hardware

It is possible to save your data on an external SD card in Arduino and to use it as datalogger. There are lots of options on the market and we decided to use the SD Card shield V4.0 of Seedstudio . Technically speaking, there is little difference to other SD card slot solutions – they are working similar.

### Datasheet

As you do it with all new components in Arduino, you first get familiar with the specs, the datasheet, the wiring and the connection to the UNO board.

SD Card shield V4.0

[Webpage: <http://wiki.seeedstudio.com/SD_Card_shield_V4.0/> ]

Let’s check the general specs such as voltage, the supported card types and which pins connect to what.

### SD Library

Arduino provides us with a library that allows us common file operations on an SD card.

<https://www.arduino.cc/en/reference/SD>

This library will need to be installed and summoned at the beginning of the sketch.

### 015 Libraries and Code

The SD card communicates with the Arduino using a protocol called SPI. Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. In fact, SPI is the default protocol for the communication with an SD card in the Arduino world.

[Webpage: <https://www.arduino.cc/en/Reference/SPI>]

[Code Example]

SPI requires the definition of four pins, three of them are common to all devices and can be shared, the fourth pin is specific to the device. In our case this means that the shared three pins are generic and connects via the ISCP header on the Arduino. We just must ensure to select the correct SS (Slave Select) pin.

By looking onto the specs of the card PIN 4 that can be used as SS(Slave Select) pin. (or CS – Chip Select). The 3 pins are default.

Now, we know the pin selection, let’s have a look at the code. To communicate with the SD card, we need to load another library called “SD Library” – this allows us to read, write and manage files on the SD card.

The steps are generally following:

* check if a communication can be established and the SD card is initialised. This is important to handle errors and to give you some feedback in case there and any problems
* Open or create a file
* Write data into the file
* Close the file

There are extra commands to remove files, create directories, delete directories but also commands that let you read files

This is almost all you need to know to use the SD card, the setup took a little bit long, but the library and the code is quite straight forward.

## COMBINED SCRIPT

The last steps is to bring everting together into one file that allows us to run the Arduino as in depended datalogging device.

It is needless to say that each datalogger is different and that the code might be adjusted and optimised. One might spend quite some time in optimising the code, and it’s always good to keep in mind that we do prototypes.

Yet the biggest issue is to handle errors that might occur during the recoding. As for an example: the Arduino script should not crash if one sensor is unplugged by accident. Or the recoding should not take place if there is no GPS signal (and the Arduino should not crash). These kinds of considerations are very important and the script has to take that into account.

### 016 Full Code

[Code Example]

## Literature

“***Exploring Arduino****”,* Jeremy Blum

This is one of the best books to learn Arduino. Professionally written and a well-balanced mix in-between electronics and code. This is the best starting point to learn Arduino and it will bring you a great way.

Whist the first book is absolutely sufficient, there are another two books that are worth mentioning if you want to dig deeper:

“***Programming Arduino: Getting Started with Sketches***”, Simon Monk

This excellent book focusses on the programming aspect of Arduino and less on the electronics. This allows Simon Monk to dig deeper into the programming aspects of Arduino: The C language is explained in depth and this book takes the reader deep into functioning of microprocessors and communication protocols.

“**Programming Arduino, Next Steps**”, Simon Monk

This book shows the reader of to manage the performance of a regular Arduino board. How can the electricity consumption be reduced? How can the memory be managed better? How can in run an Arduino as a super-efficient datalogger for periods than more than a year.