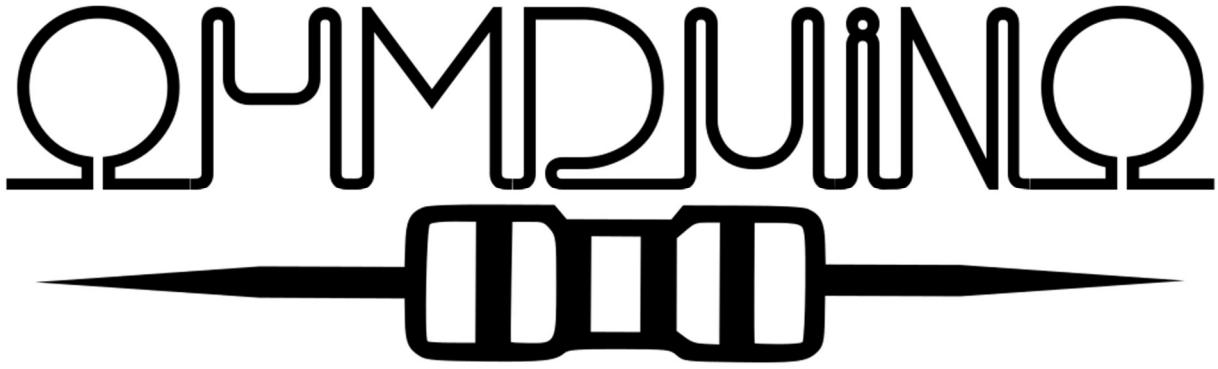
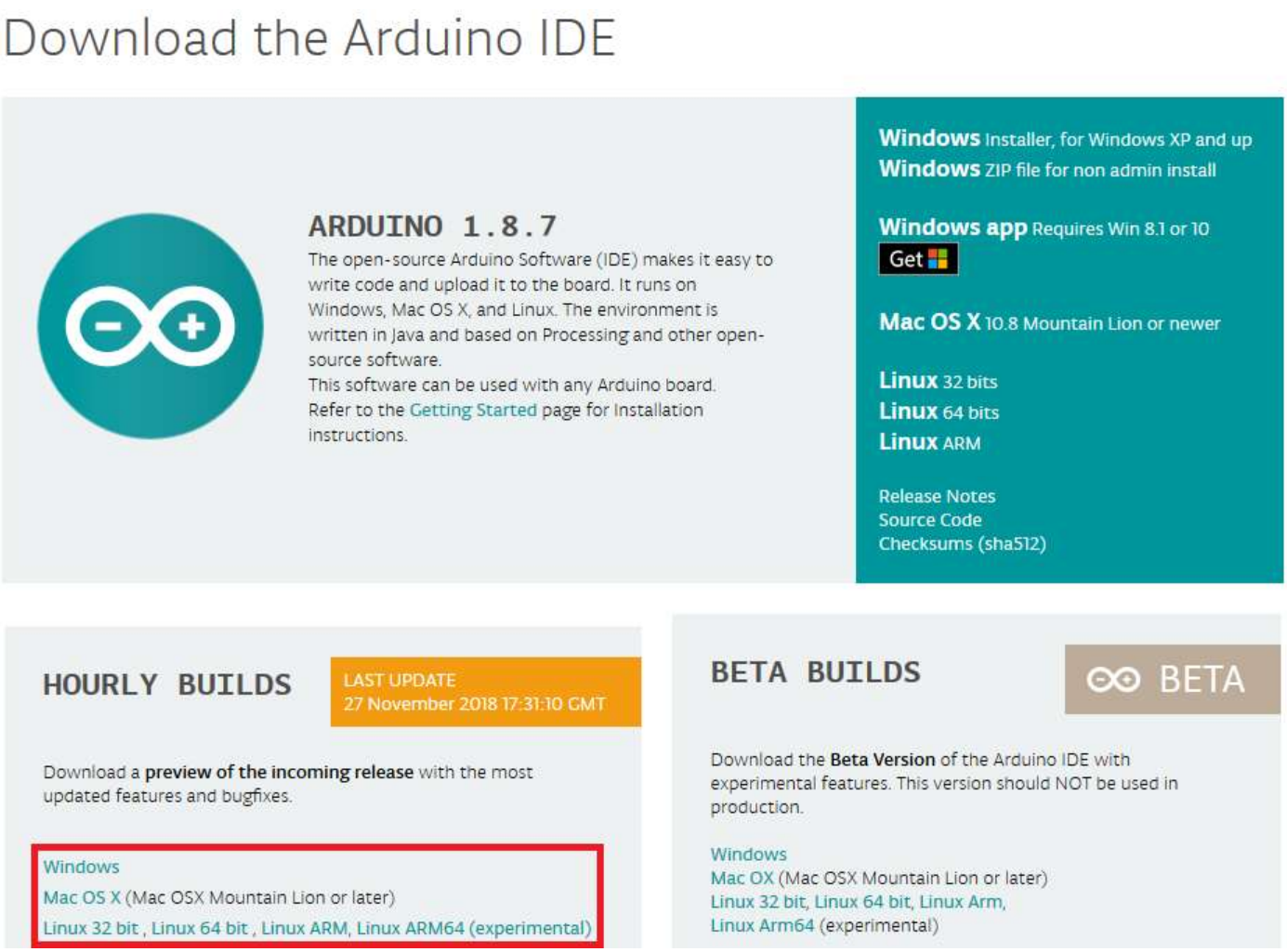


KEY TO BITS AND BYTES

Software

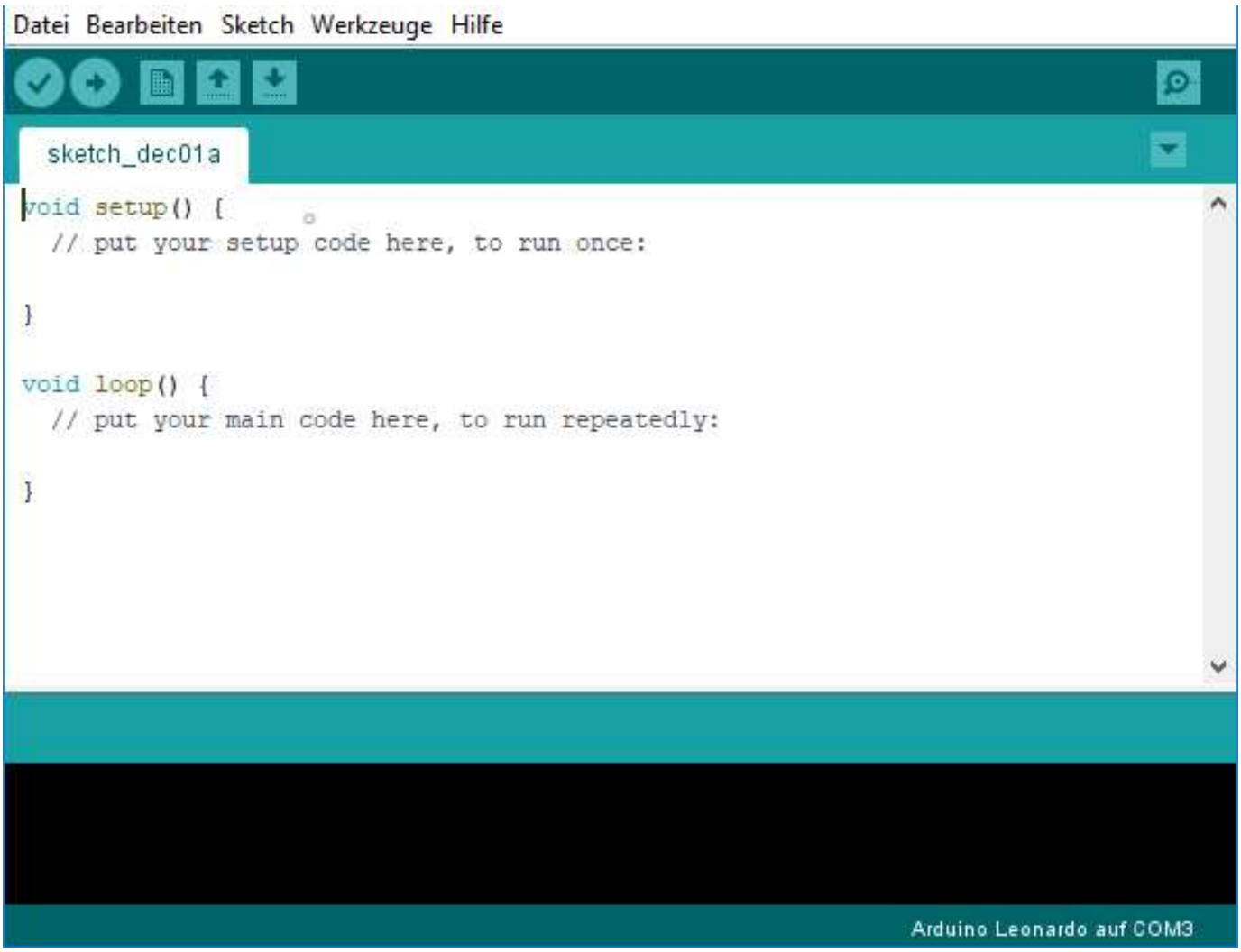
Herzlichen Glückwunsch! Du hältst nun den „Key to Bits and Bytes“ in deinen Händen. Bevor wir allerdings mit dem Einstieg in die Welt der sogenannten Programmiersprache C++ beginnen, musst du dir erst die geeignete Software zu eigen machen. Unter dem folgenden Link kann diese (natürlich kostenlos) herunterladen werden:

https://www.arduino.cc/en/Main/Software



Im rot markierten Bereich einfach dein verwendetes Betriebssystem auswählen, die Daten herunterladen und den Installationsanweisungen folgen.

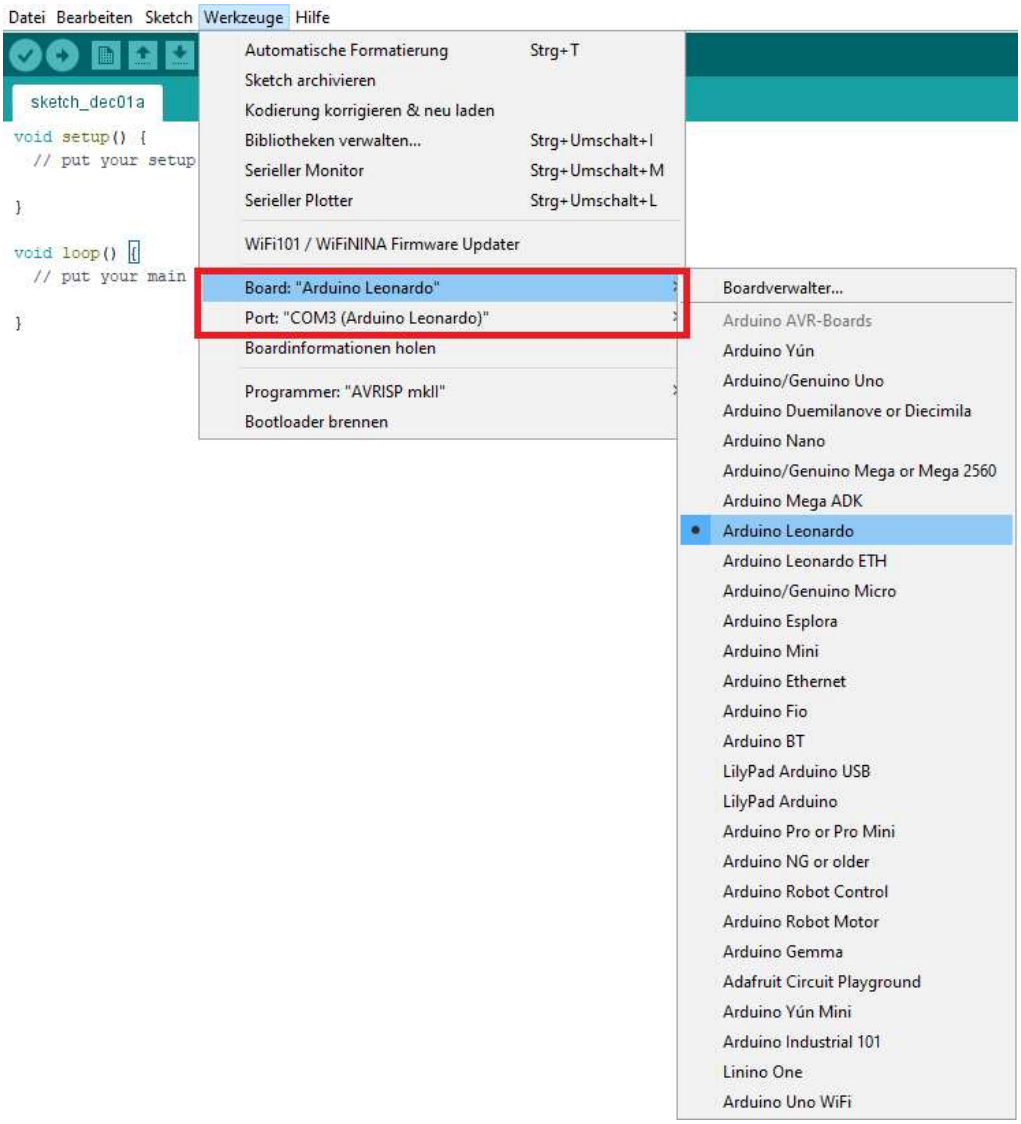
Wenn die Installation erfolgreich abgelaufen ist, kannst du das Programm ausführen und solltest nun folgendes Fenster vor dir haben:



In dem Fenster sind schon die ersten Zeilen unseres zukünftigen Programms vorgefertigt. Wahrscheinlich sehen diese noch wie Hieroglyphen für dich aus. Dies soll dich jetzt aber nicht entmutigen. In ein paar Schritten wirst du damit spielend leicht umgehen können.

Der beliebteste Fehler

Um den beliebtesten aller Fehler bereits vorzubeugen, musst du nun deinen frisch erworbenen OhmDuino in einen freien USB-Port an deinen Computer stecken. Falls schon ein Programm darauf vorhanden ist, werden bereits jetzt ein paar LEDs zu leuchten beginnen – also nicht wundern. Nun müssen wir an der Software ein paar kleine Änderungen vornehmen:



In der **Menüleiste** unter **Werkzeuge** findest du die Menüreiter **Board** und **Port**. Die vorzunehmenden Einstellungen sind im obigen Bild rot markiert:

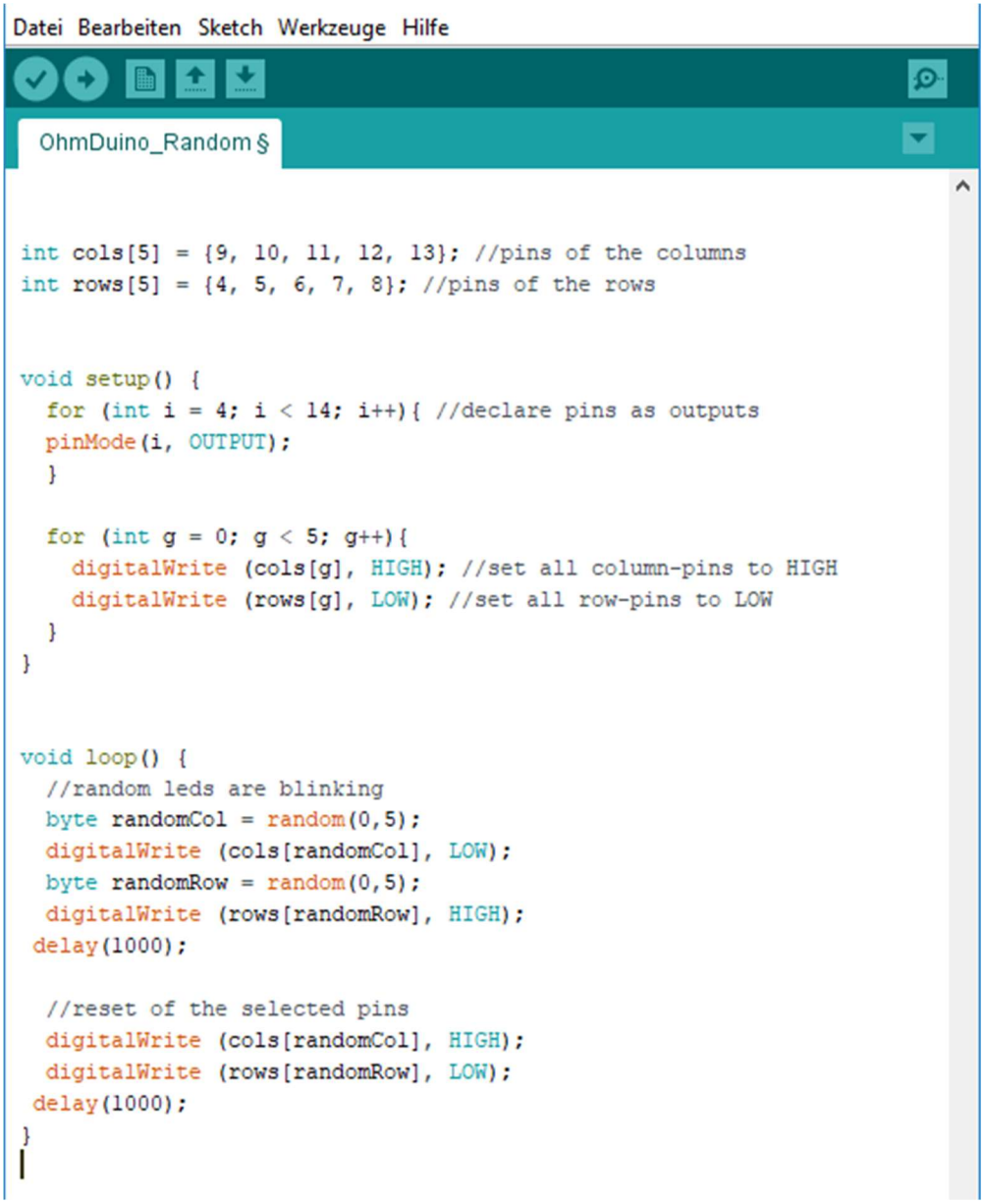
- **Board: „Arduino Leonardo“**
- **Port: „COM3 (Arduino Leonardo)“**

Nur wenn dies richtig ausgewählt wurde, kannst du auf deinen OhmDuino zugreifen.

Ein richtiges Programm

Anhand der Analyse eines richtigen Programmes, lernt man das Programmieren am leichtesten. Man kann von vorne bis hinten alle Bestandteile besprechen. Somit wird der roten Faden nie aus den Augen gelassen und die Lernphase ist möglichst praxisnah.

Das folgende Programm steuert immer eine zufällige LED auf unsrem OhmDuino für eine volle Sekunde an:



Eine wilde Aneinanderreihung von Zahlen, Buchstaben und Symbolen. Sehr beeindruckend, nicht wahr?

Wenn du an dieser Stelle schon dein erstes Erfolgserlebnis haben möchtest, kannst du das Programm gerne abtippen und ausprobieren. Allerdings lernt man beim sturen Kopieren nicht sehr viel. Darum werden wir zuerst Schritt für Schritt die Programmzeilen durchsprechen. Grundlegend lässt sich die Programmstruktur eines C-Programms in drei Teile aufteilen.

Deklaration der Variablen (1. Teil)

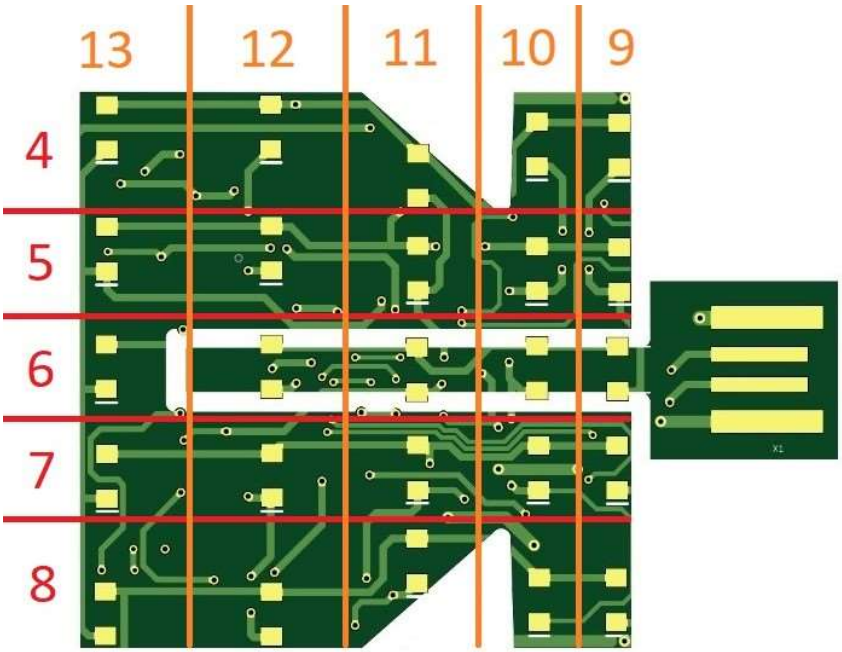
Variablen kennen wir alle aus der Mathematik. Das Paradebeispiel dafür ist X. Wie wir alle wissen, kann X für jede erdenkliche Zahl stehen. Kommt drauf an, in welcher mathematischen Gleichung es sich versteckt hat.

So ähnlich ist es auch beim Programmieren. Man hantiert mit Variablen, die einfach für Zahlen stehen. Anstatt der mathematischen Gleichung kommt es hierbei allerdings auf das Programm an, in dem sich die Variable versteckt hat. Der Unterschied ist nur, dass wir vor Verwendung der Variable dem Programm sagen müssen, in welchem Bereich die verwendete Zahl zu finden ist. Das wird dann in der Fachsprache Deklaration genannt.

```
int cols[5] = {9, 10, 11, 12, 13}; //pins of the columns
int rows[5] = {4, 5, 6, 7, 8}; //pins of the rows
```

Am Anfang unseres Programms finden wir zwei Zeilen, die mit **int** beginnen. **int** steht für Integer (ganze Zahl). Das bedeutet, die folgende Variable kann Werte zwischen -32.786 und +32.767 annehmen. Auf gut Deutsch: Eine stinknormale Zahl.

Nach einem Leerzeichen stehen jeweils vier Buchstaben in unseren Programmzeilen. Diese Buchstaben bilden die Namen unserer Variablen, die deklariert wurden. Diese heißen aber nicht X oder Y, sondern **cols** und **rows**. Das sind englische Abkürzungen für „Spalten“ (columns) und „Reihen“ (rows). Die Namen dieser Variablen wurden nicht ohne Grund so gewählt. Um das zu verstehen, müssen wir uns die Hardware auf unserem OhmDuino kurz ansehen:



Die 25 LEDs auf dem OhmDuino sind in Gruppen verschaltet. Die roten Gruppen (4, 5, 6, 7, 8) entsprechen unseren Reihen. Das bedeutet, dass die +-Pole der jeweiligen LEDs in diesen Reihen miteinander verbunden sind. Die orangenen Gruppen (9, 10, 11, 12 ,13) entsprechen unseren Spalten. Dort sind jeweils alle –-Pole miteinander verbunden. Um eine LED zum Leuchten zu bringen, muss man ihr am +-Pol ein HIGH- und am –-Pol ein LOW-Wert zuweisen. Aber dazu später mehr.

Zurück zu unserer Deklaration der Variablen:
Gleich nach den Namen unserer Variablen befindet sich eine **[5]**. Diese **[5]** macht aus einfachen Variablen ein sogenanntes Array. Dadurch ist es möglich nicht nur einen Wert zu speichern, sondern mehrere. Die **5** in eckigen Klammern hinter dem Variablennamen bestimmt die Anzahl der Speicherplätze. Somit haben wir aus einer einzigen Variablen quasi fünf gemacht. Die darauffolgenden **= {9, 10, 11, 12, 13}** bzw. **= {4, 5, 6, 7, 8}** weißen den Arrays, also unseren jeweils fünf Speicherplätzen, noch jeweils eine Nummer zu. Anhand dieser Nummer werden wir die Speicherplätze später noch auseinanderhalten bzw. verwenden können. Wem es noch nicht aufgefallen ist, die Zahlen in den geschweiften Klammern entsprechen genau den Zahlen aus der OhmDuino LED-Gruppenverteilung.

Am Ende jeder Programmierzeile muss ein **;** (Semikolon) stehen. Wundere dich nicht darüber, akzeptiere es einfach. Wenn du einmal nicht daran denkst, wird das Programm beim Übertragen sofort eine Fehlermeldung anzeigen. Spätestens dann wirst du es akzeptieren müssen.

Du fragst dich bestimmt, warum jetzt schon die Rede vom Ende der Programmzeile war. Ganz einfach, denn alles war hinter einem **//** (doubleslash) steht, wirkt nicht in das Programm mit ein. Das ist der Bereich für die Kommentare des Programmierers. Du solltest dein Programm immer mit Kommentaren versehen. Wenn du einmal mehrere hundert Zeilen verfasst hast, verlierst du sonst schnell den Überblick. Dann kennst du dich selbst nicht mehr aus und eine dritte Person schon gar nicht. Daher denk an diese Worte: „Das Programm immer mit Kommentaren versehen!“

Die ersten zwei Zeilen hast du nun überstanden. Keine Sorge, dass waren die Schwierigsten. Falls du nicht alles verstanden hast, ist es auch nicht schlimm. Für die Programmierung auf unserem OhmDuino kannst du diese Zeilen immer übernehmen. Es sollte dir nur einmal erklären werden, was dahintersteckt.

Void Setup (2. Teil)

Hier werden Grundeinstellungen (z.B. ob ein Kanal ein In- oder Output ist) vorgenommen. Dieser Programmbereich wird nur beim Programmstart ausgeführt, also genau ein Mal.

```
void setup() {
  for (int i = 4; i < 14; i++){ //declare pins as outputs
  pinMode(i, OUTPUT);
  }

  for (int g = 0; g < 5; g++){
    digitalWrite (cols[g], HIGH); //set all column-pins to HIGH
    digitalWrite (rows[g], LOW); //set all row-pins to LOW
  }
}
```

Die erste Zeile sagt aus, dass alles zwischen den äußersten geschweiften Klammern **{}** sich im **void setup** befindet. Die runden Klammer **()** solltest du für den Anfang einfach mitführen. Ihre Funktion würde den Rahmen dieser Kurzanleitung sprengen.

In der zweiten Zeile bedienen wir uns einer sogenannten **for**-Schleife. Als Parameter werden in den Klammern die Initialisierung (**int i = 4**), die Abbruchbedingung (**i < 14**) und die Fortsetzung (**i++**) übergeben. Die Anweisung unserer Schleife steht in der dritten Zeile **pinMode(i, OUTPUT)**. Unsere **for**-Schleife zählt also die Zahlen von **4** bis **i <14** (also 13) hoch. Jede einzelne Zahl wird durch die angegebene Anweisung **pinMode(i, OUTPUT)** als Ausgang gesetzt. Somit ist es uns nun möglich, an den Kontakten unserer LED-Gruppen Signale auszugeben.

In unserem **void setup** wird noch mit einer zweiten **for**-Schleife gearbeitet. Genauso wie vorher gibt es wieder eine Initialisierung (**int g = 0**), eine Abbruchbedingung (**g < 5**) und die Fortsetzung (**g++**). Dieses Mal werden je zwei Anweisungen verfolgt **digitalWrite (cols[g], HIGH)** und **digitalWrite (rows[g], LOW)**. Diese **for**-Schleife zählt die Speicherplätze unserer am Anfang deklarierten Variablen (**cols** und **rows**) durch. Weiterhin weißt sie den Speicherplätzen der **cols** den Wert **HIGH** (5 Volt) zu und den Speicherplätzen der **rows** den Wert **LOW** (Masse) zu.

Am Ende des **void setup** können wir also durch unser Programm auf die Ausgänge der LED-Gruppen zugreifen. Des Weiteren wurde allen +-Polen der LEDs ein **LOW**-Wert und den –-Polen ein **HIGH**-Wert zugewiesen. Somit sind die perfekten Bedingungen für eine LED-Ansteuerung gegeben.

Void Loop (3. Teil)

Der **void loop** wird im Gegensatz zum **void setup** ständig wiederholt. Hier wird der eigentliche Programmablauf geschrieben. Um ehrlich zu sein, ist das der spaßigste Teil des Programms. Wenn man Änderungen vornimmt, können diese an den LEDs direkt überprüfen. Damit ist alles leichter nachvollziehbar und man kann ein bisschen mit seinem OhmDuino herumspielen.

```
void loop() {
  //random leds are blinking
  byte randomCol = random(0,5);
  digitalWrite (cols[randomCol], LOW);
  byte randomRow = random(0,5);
  digitalWrite (rows[randomRow], HIGH);
 delay(1000);

  //reset of the selected pins
  digitalWrite (cols[randomCol], HIGH);
  digitalWrite (rows[randomRow], LOW);
 delay(1000);
}
```

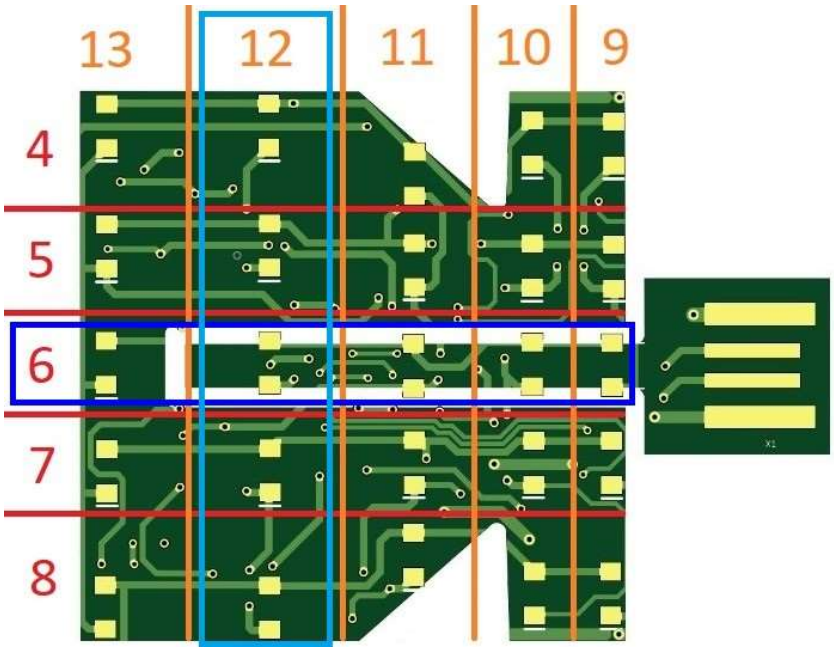
Die erste Zeile sagt wieder aus, dass wir uns im **void loop** befinden. Genauso wie beim **void setup** wird der Bereich des **void loops** mit den äußersten geschweiften Klammern **{}** gekennzeichnet. Auch hier solltest du die runden Klammern **()** einfach mitführen.

In diesem Programmteil befindet sich in der zweiten Zeile ausschließlich der Kommentar. Nicht wie in den vorherigen Teilen, wo beinahe jede Programmzeile am Ende einzeln kommentiert wurde. Das hat folgenden Grund: In diesem Programmteil beschreibt der Kommentar nicht nur eine Zeile, sondern einen ganzen Programmabsatz. Der Kommentar dient sozusagen als Überschrift für das darauffolgende.

In der dritten Zeile starten wir jetzt mit unserem eigentlichen Programm. Über den Befehl **byte randomCol** legen wir wieder ein Parameter fest. Dieser Parameter erhält den Namen **randomCol** (zufällige Spalte). Darin kann ein Wert gespeichert werden, der ein **byte** (Zahl zwischen 0 und 255) nicht überschreitet. Dahinter sagen wir **= random(0,5)** was uns erlaubt, bei jedem Programmablauf eine zufällige Zahl zwischen **0** und **5** für unseren Parameter festzulegen.

Die darauffolgende Zeile verwendet unsere im 1. Teil deklarierte Variable **cols** und ändert einen der fünf Speicherplätze von **HIGH** zu **LOW**. Wir erinnern uns zurück: die Variablen **cols** und **rows** wurden als Array festgelegt. Daher stellen sie fünf Speicherplätze zur Verfügung. Wir schreiben jetzt einfach in einen der Speicherplätze über den Befehl **digitalWrite** den Wert **LOW**. Welcher der fünf Speicherplätze das sein wird, wissen wir selbst nicht. Wir haben es ja als Zufall programmiert. Zur besseren Vorstellung wird es noch mal hardwaremäßig erklärt: Wir haben in eine der fünf Spalten unserer LED-Gruppen (das waren die Orangenen: 9, 10, 11, 12, 13) auf unserem OhmDuino das Signal an unseren –-Polen von **HIGH** (5 Volt) zu **LOW** (Ground) geändert.

Damit jetzt noch eine der LEDs zu leuchten beginnt, muss ihr am +-Pol ein **HIGH**-Signal ausgegeben werden. Dafür sorgen die Zeilen fünf und sechs. Sie sind genauso aufgebaut wie die Zeilen drei und vier. Nur dieses Mal wird ein Parameter namens **randomRow** generiert, es wird eine LED-Gruppe der Reihen ausgewählt (das waren die Roten: 4, 5, 6, 7, 8) und der Wert wird von **LOW** zu **HIGH** geändert.



Hier ein Beispiel an unserem OhmDuino:
Laut dem Bild wird der Reihe 6 ein **HIGH**-Wert zugewiesen (dunkelblau dargestellt). Das heißt die Zahl die uns zufällig in der dritten Programmzeile generiert wurde ist die 3. Warum die 3? Ganz einfach, die sechste Reihe entspricht dem 3. Speicherplatz unseres **rows**-Arrays. Vielleicht wird das Ganze bei der Betrachtung der Spalten klarer. Der Spalte 12 wird ein **LOW**-Wert zugewiesen (türkis dargestellt). Somit wurde uns in der fünften Programmierzeile zufällig die Zahl 4 generiert. Denn die Spalte 12 entspricht dem 4. Speicherplatz unseres **cols**-Arrays. Dort, wo sich der dunkelblaue und der türkise Rahmen überschneiden, hat die LED an dem +-Pol einen **HIGH**-Wert und am –-Pol einen **LOW**-Wert. Diese LED ist also die Glückliche, die aufleuchten wird.

Herzlichen Glückwunsch! Würdest du das Programm bis dahin abtippen, würden deine ersten selbstprogrammierten LEDs zu leuchten beginnen. Leider würden die LEDs sich nicht mehr ausschalten lassen. Der eigentliche Zweck ist damit knapp verfehlt. Deshalb werden wir uns noch die letzten paar Zeilen anschauen müssen.

Die nächste Zeile **delay (1000)** ist sowohl leicht zu programmieren, als auch leicht zu verstehen. **delay** heißt ins Deutsche übersetzt Verzögerung. Die Zahl in Klammer steht für Millisekunden. Zusammen bedeutet das so viel wie: 1 Sekunde warten! Diese Verzögerung sorgt dafür, dass unsere angesteuerte LED am Ende genau **1000** Millisekunden leuchten wird.

Kommen wir noch zu dem Erlöschen der LEDs. Im Grunde genommen ist es ganz einfach. Wir setzen einfach alle Speicherplätze von unseren **cols**- und **rows**-Arrays auf den ursprünglichen Wert. Dazu verwenden wir wieder den Befehl **digitalWrite**. Das heißt, wir setzen alle Spalten unserer LED-Gruppen (das waren die Orangenen: 9, 10, 11, 12, 13) auf den Wert **HIGH** und die LED-Gruppen der Reihen (das waren die Roten: 4, 5, 6, 7, 8) auf den Wert **LOW**. Am Ende wurde noch ein **delay**

gesetzt. Dadurch wird **1000** Millisekunden lang keine LED angesteuert. Die beiden verwendeten **delay**-Befehle sorgen dafür, dass eine zufällige LED **1000** Millisekunden angesteuert wird und danach **1000** Millisekunden keine LED leuchtet. Wir haben somit einen Blinktakt programmiert.

Nach unserem **void loop** ist das Programm auf demselben Stand wie vorher. Dadurch kann es unendlich oft abgearbeitet werden. Es wird bei jedem Programmablauf eine zufällige LED aufleuchten, weil wir durch den Befehl **random** immer eine zufällige Reihe bzw. eine zufällige Spalte ansteuern. Am Ende werden alle Signale wieder auf den Anfangswert zurückgesetzt. Daher ist es egal, welche LED vorher angesteuert wurde.

Wir sind fertig mit deinem ersten Programm. Hoffentlich hattest du Spaß. Im nächsten Teil darfst du selbst endlich mit Programmieren loslegen.

Aller Anfang ist schwer

Wenn du deinen OhmDuino an deinen Computer angesteckt hast, dem „beliebtesten Fehler“ zuvorgekommen bist und ein Programm vorweisen kann, solltest du in der Lage sein dieses Programm zu übertragen.

Mit dem rot markierten Knopf kannst du deine Programme auf den OhmDuino übertragen. Für den Anfang solltest du dich ein wenig mit unserem Beispiel-Programm auseinandersetzen. Ändere ein paar Zahlen! Lösche ein paar Zeilen raus! Füge selbst ein paar neue Zeilen hinzu! Deiner Kreativität sind keine Grenzen gesetzt. Falls am Ende gar nichts mehr aufleuchten sollte, kannst du das Programm einfach wieder in den Urzustand bringen. Durch das Herumspielen lernt man am schnellsten und vor allem ist man mit Motivation dabei.

Auf folgender Internetseite findest du noch weitere von uns geschriebene Programme (diese Art von Programmen werden übrigens auch „Sketch“ genannt):

https://hackaday.io/project/162242-ohmduino

Falls du am Anfang noch Schwierigkeiten hast deine eigenen Sketche zu schreiben, kannst du gerne unsere Vorgefertigten herunterlanden. Wenn du mit den verschiedenen Sketchen ein wenig herumspielst, gewöhnst du dich langsam an die Denkweise des C-Programmierens. Außerdem kannst du dir ein paar Inspirationen für deine persönliche Ohmduino-LED-Ansteuerung holen.

Das Wichtigste beim C-Programmieren sind die verschiedenen Befehle. Natürlich gibt es noch viel mehr als in unserem Beispiel-Programm verwendet wurden. Ein schönes Dokument mit den wichtigsten Befehlen findest du unter:

http://www.schwarze-schuette.de/wp-content/uploads/2011/05/00b_Arduino-Befehlsu%CC%88bersicht.pdf

Dort wurde zu den Befehlen noch eine kurze Beschreibung mit ausgeführt.
Du kannst auch einfach mal danach googeln. Das Internet ist voll mit Informationen zur C-Programmierung.

Wichtige Sicherheitshinweise

ACHTUNG:
Beim Umgang mit Elektrogeräten ist stets Vorsicht geboten, u. a. sind die folgenden grundsätzlichen Hinweise zu beachten:

- Vor Benutzung des Geräts alle Anweisungen lesen.
- Zur Verringerung der Verletzungsgefahr Elektrogeräte in Anwesenheit von Kindern nur unter Aufsicht betreiben.
- Nur vom Hersteller empfohlenes oder verkauftes Zubehör verwenden.
- Nicht im Freien betreiben.
- Zur Verringerung der Elektroschockgefahr wieder aufladbare Komponenten von Wasser oder sonstigen Flüssigkeiten fernhalten. Elektrogeräte so platzieren, dass sie nicht in Badewanne, Wasch- oder Spülbecken fallen bzw. gezogen werden können.
- Die Verwendung des OhmDuinos in Verbindung mit Kleidung aus polyesterartigen Kunststoffen kann unter Umständen zur Zerstörung des Produktes führen.