First Step by Step to Arduino

Die Angst vor dem Unbekannten überwinden.

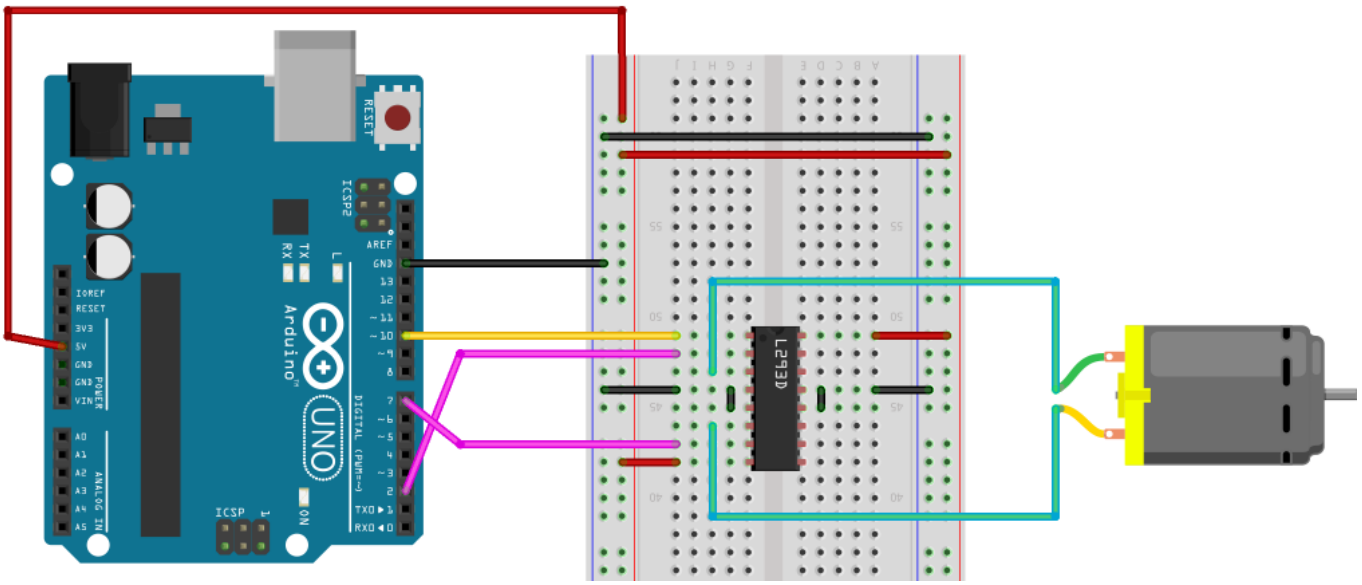
Was ist ein Arduino:

- ist eine physical computing platform
- ist open source
- besticht durch seine einfachen Konzepte
- hat eine RIESEN Community (das bietet Chancen und Risiken)
- unterstützt eine Vielzahl von Boards

Ein Arduino wird eingesetzt um Dinge in der greifbaren Welt mit einem Computer zu steuern oder aber Sensoren als Ein- und Aus-gabegeraete zu nutzen.

Ursprünglich wurde der Arduino für Kunststudierende entwickelt.

- (Kunst)Installationen
- Einführung in die Elektronik
- Prototyping
- Simple Robotik
- Ausprobieren, Lernen, Basteln



1. Arduino IDE

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

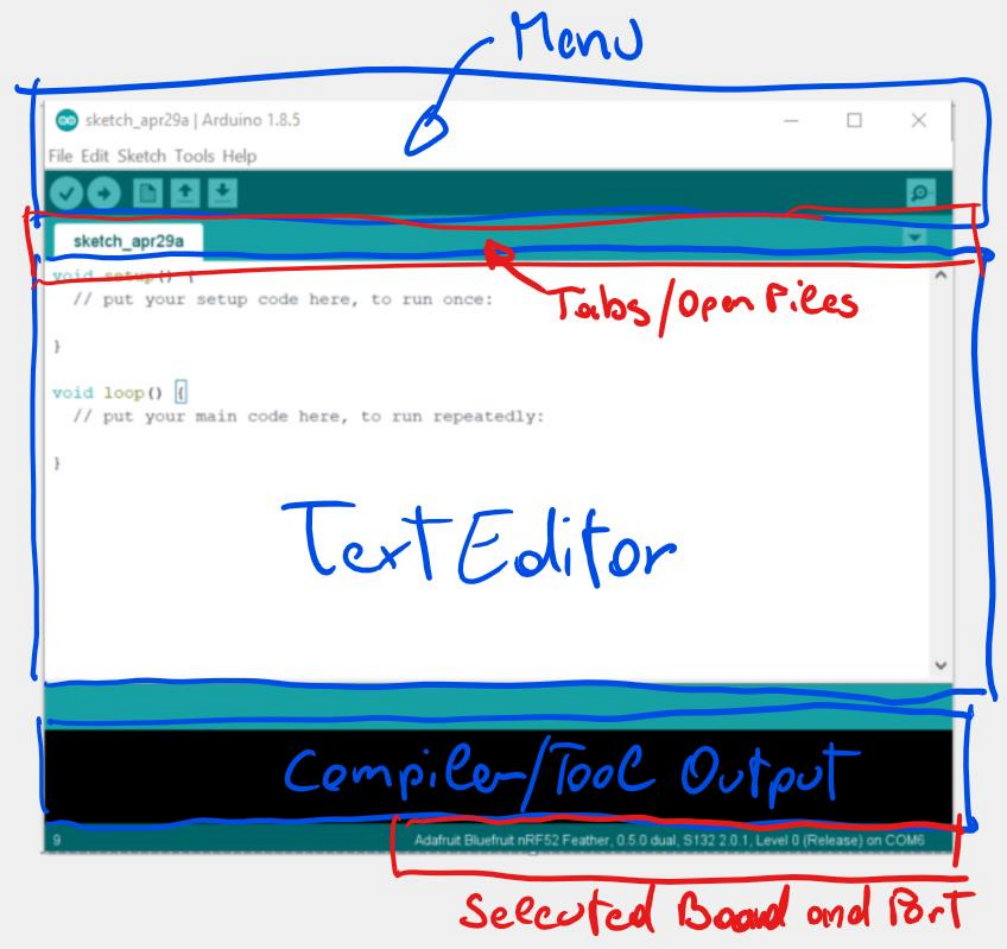
History: https://de.wikipedia.org/wiki/Arduino_(Plattform)

1.1 Installieren der IDE

https://www.arduino.cc/en/Guide/HomePage

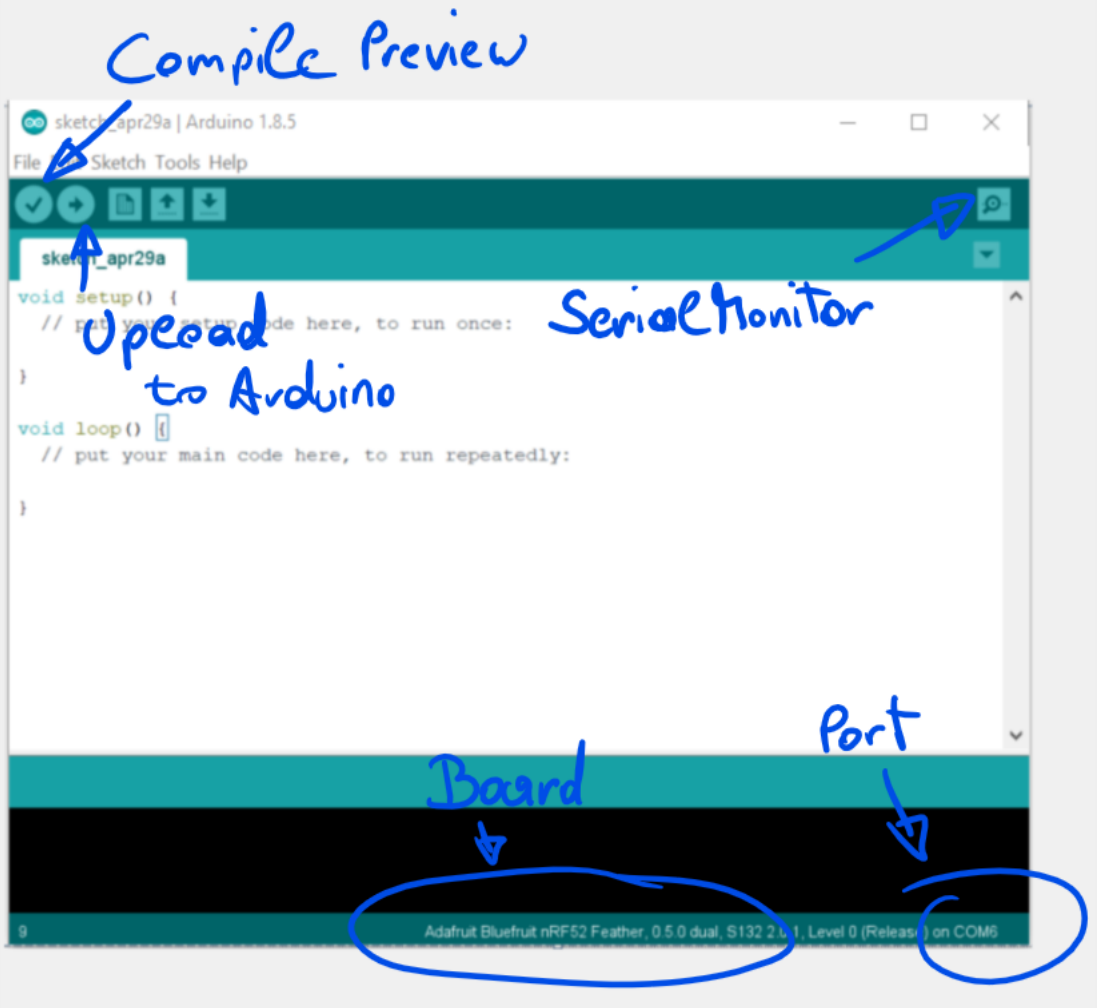
1.2 IDE

Mit der Arduino IDE (integrated development environment) kann SourceCode geschrieben, kompiliert und auf den Arduino übertragen werden. Die Arduino IDE bietet im Vergleich zu den meisten anderen IDEs oder Texteditors nicht viel Komfort und Funktionen, aber sie reicht aus um alle Programme damit erstellen und kompilieren zu können.



Die Sprache mit der Programmiert wird ist C/C++ welche vor dem Kompilieren mit dem Präprozessor „aufbereitet“ wird. Dieser Präprozessor ermöglicht es bequemer Code zuschreiben bzw. erlaubt SourceCode zu kompilieren der normalerweise so nicht kompilieren würde.

1.3 IDE Button-Funktionen



Compile:
Mit Compile wird der SourceCode (Text) zu Maschinen Code umgewandelt und ein binäres Programfile wird erzeugt. Der Compiler kann nur SourceCode ohne semantische und syntaktische Fehler kompilieren. Besteht ein Fehler wird der Fehler und die Zeile in welcher der Fehler besteht im Compiler-Output ausgegeben.

Upload:
Mit der Upload-funktion wird ein kompiliertes Programm auf den Arduino übertragen.
Hierfür muss das Arduinoboard sowie das Serielle Port richtig ausgewählt sein (siehe „1.4 Arduino Board auswählen“ sowie „1.5 Auswahl der Seriellen Schnittstelle/Port“).
Nach erfolgreichen Upload startet das Programm automatisch am Arduino.

In der unteren Programmleiste werden das aktuell ausgewählte Arduino Board sowie die Serielle Schnittstelle angezeigt.

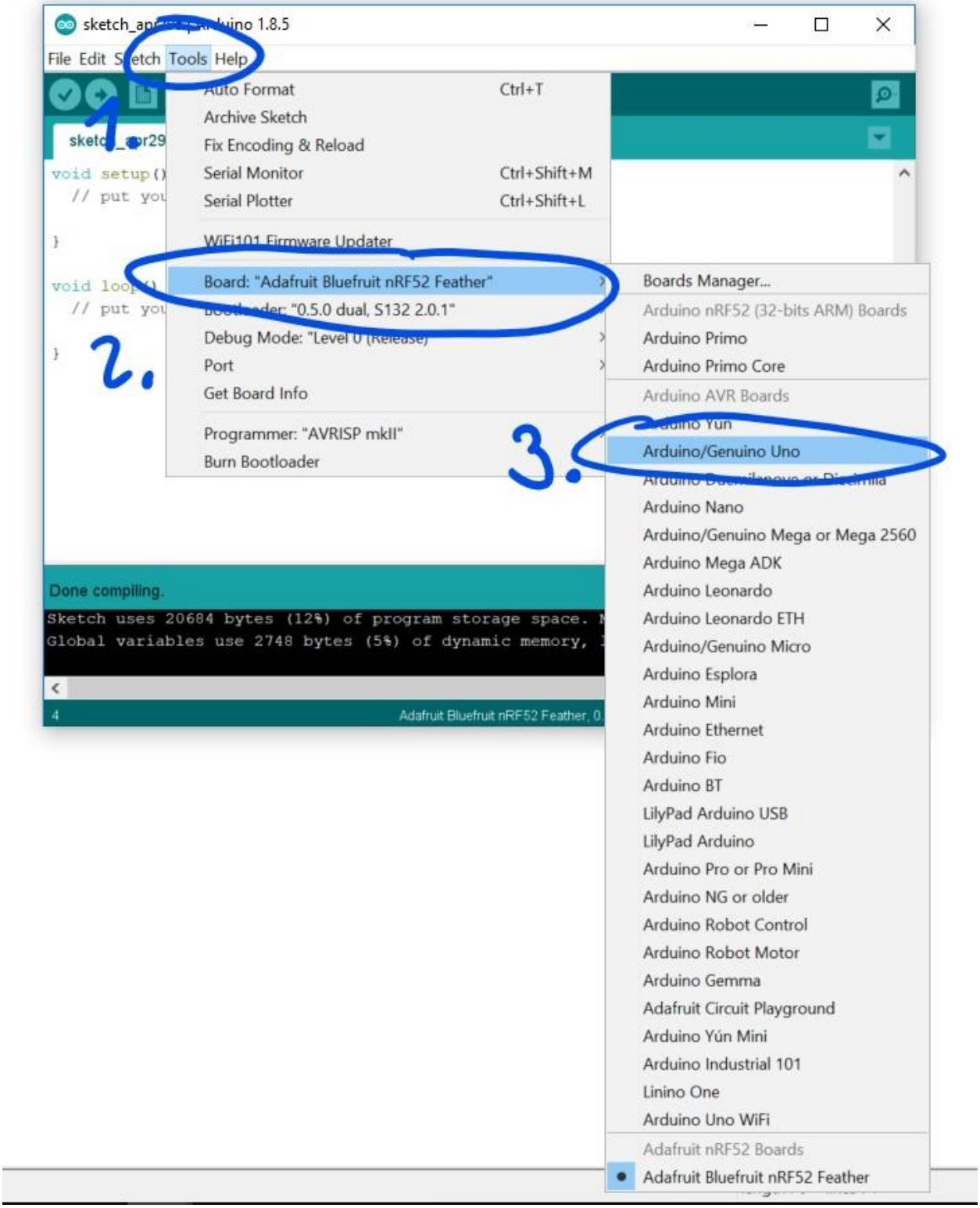
Serial Monitor:
Mit dem Serial Monitor Button kann der Serial Monitor geöffnet werden.

1.4 Arduino Board auswählen

Installierte Arduino Boards können mit Tools->Board-> ausgewählt werden.

Weiter Arduino Boards können mit dem Boards Manager hinzugefügt werden.

https://www.arduino.cc/en/Guide/Cores

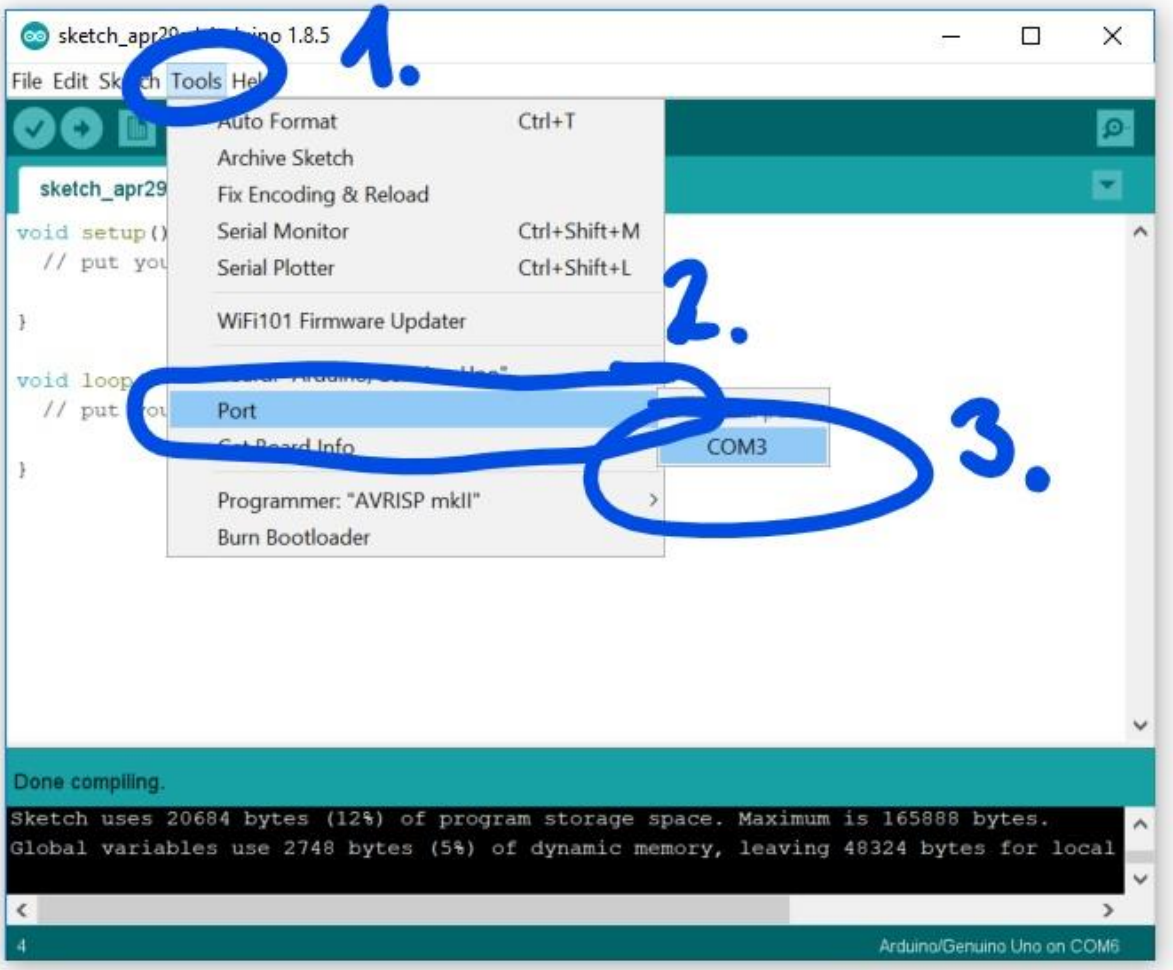


1.5 Auswahl der Seriellen Schnittstelle/Port

Man muss die Serielle Schnittstelle mit Tools->Port-> auswählen.

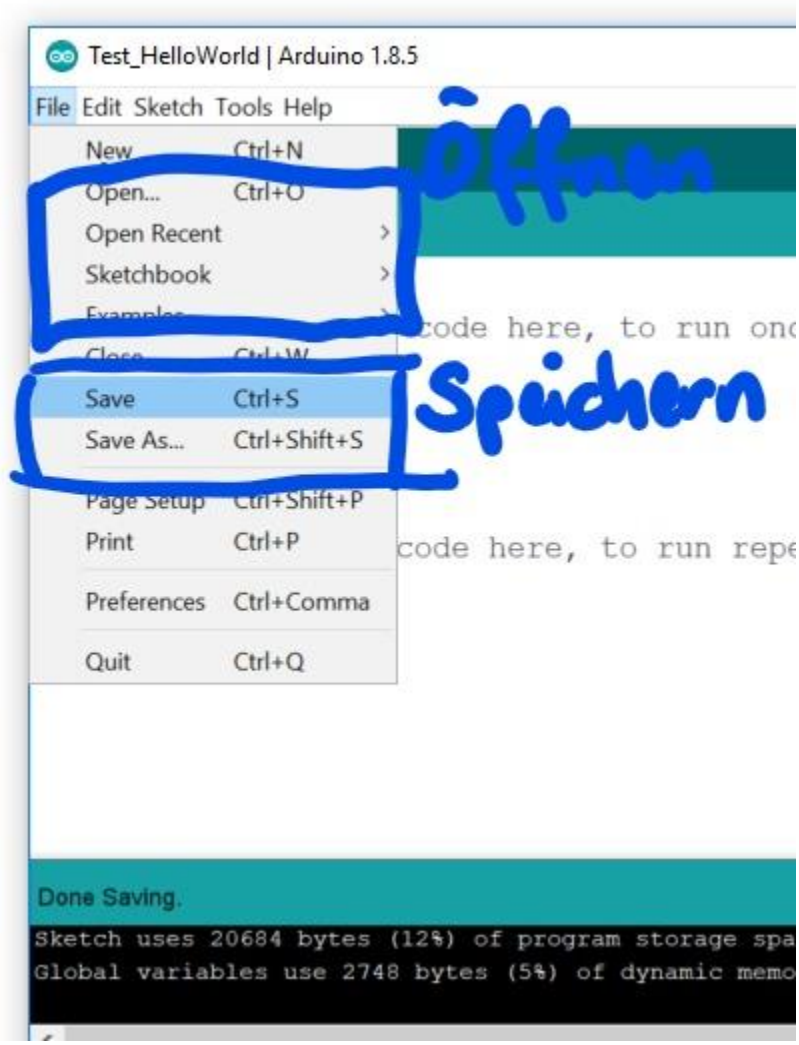
Wenn man sich nicht sicher ist welche Schnittstelle der Arduino ist, kann dies am schnellsten durch an und abstecken rausgefunden werden:

- Arduino abstecken
- Tools->Port-> Liste merken
- Arduino anstecken
- Tools->Port-> sehen welches Port neu hinzugekommen ist.
  ! Wichtig ist das die Menu-Tools-Dropdown Auswahl dazwischen geschlossen war damit die Port List aktualisiert wird.



1.6 Öffnen und Speichern

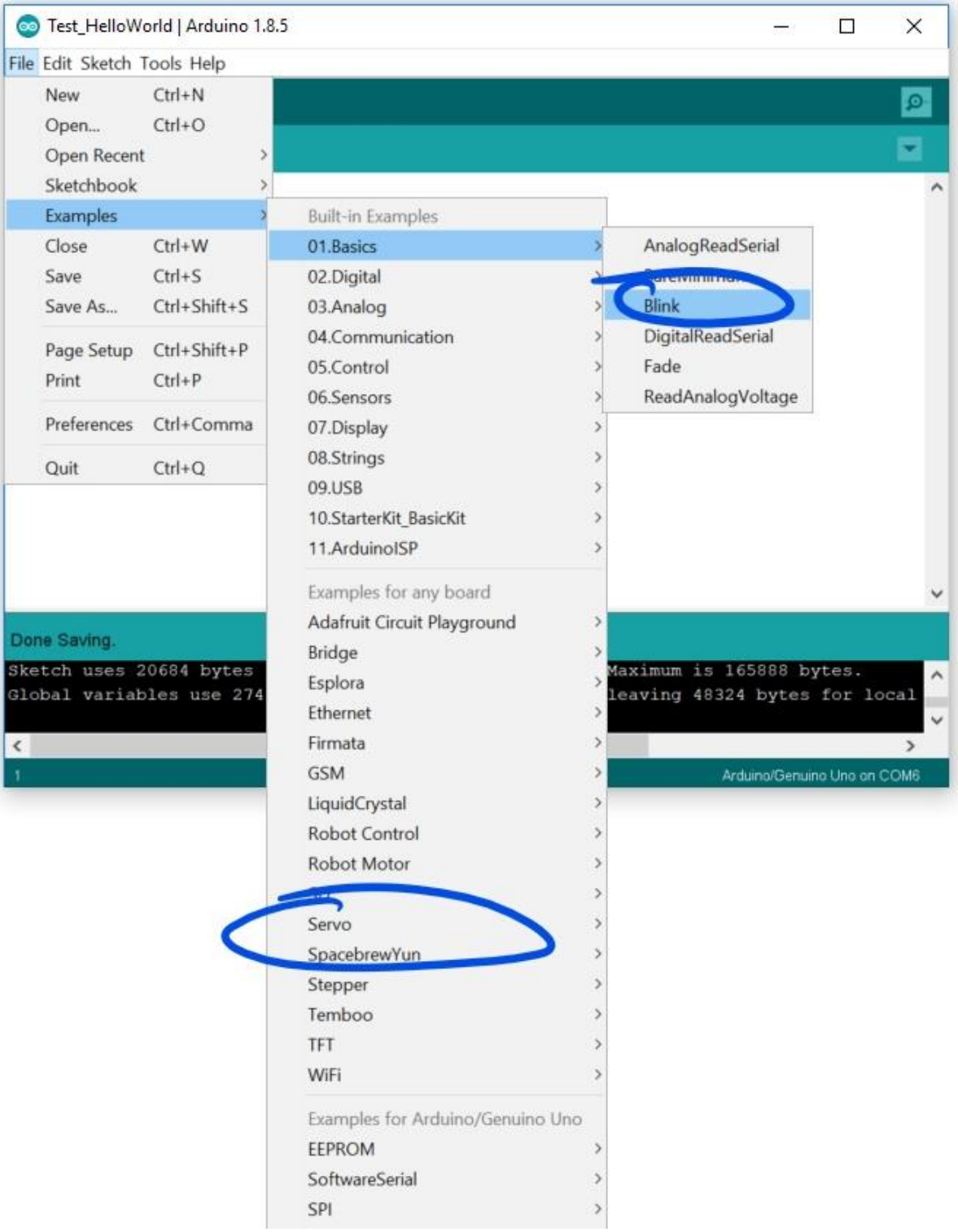
Wenn ein Projekt gespeichert wird kann man dieses im Sketchbook finden und wieder öffnen.



1.7 Example Code

Es gibt viele Beispiel Programme die per File->Examples->...->... geöffnet werden können.

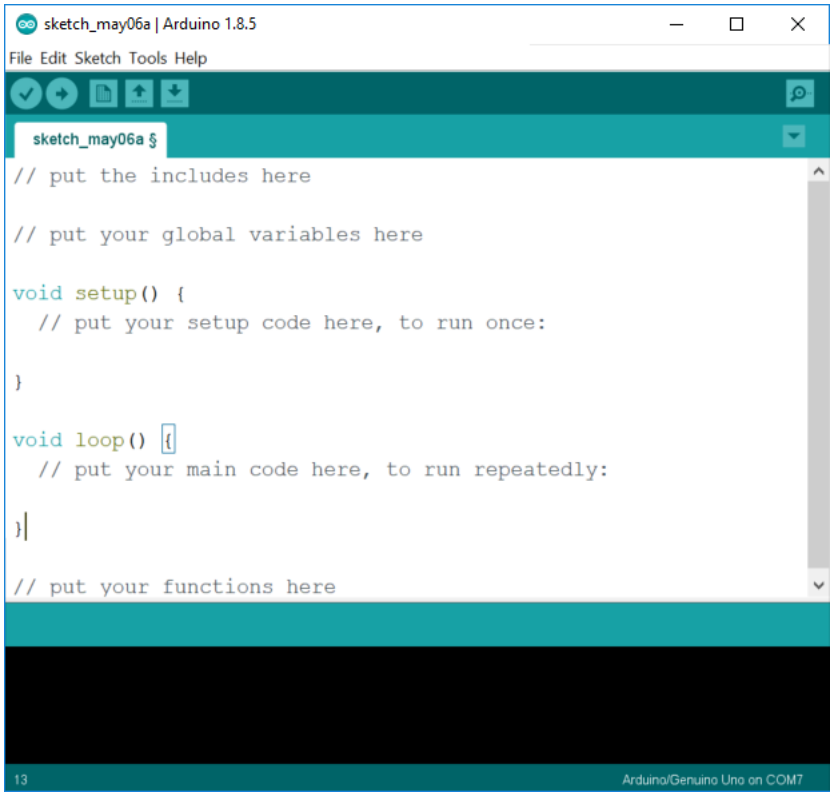
Diese SourceCode Beispiele sind sehr hilfreich um einen schnellen Einblick in das Arbeiten/ansteuern von Sensoren/Displayes/Motor/Schnittstellen/etc... zubekommen.
Man findet für die meisten DIY-Elektronik ArduinoLibraries und Example Code und es kann auch einfach Example Code selbst hinzugefügt werden. https://www.arduino.cc/en/Tutorial/LibraryExamples



2. Arduino Program-Flow

Ein Arduino Programm hat eine setup Funktion die zum Start des Programmes aufgerufen wird.
Man benützt sie zum Initialisieren der Hardware und des Programmes.

Die loop Funktion wird nach dem die setup Funktion fertig ist permanent in einem loop aufgerufen. Ist die loop Funktion fertig wird sie gleich wieder aufgerufen.
Man benützt sie für das eigentliche Prgrammvorhaben.



Libraries inkludiert man am Anfang des Programcodes (ganz oben).

```
#include<Servo.h>
```

Globale Variablen/Objekte sollte man direkt nach den Include-Statements definieren.

```
Servo yourServoX;         // Servo Objekt yourServoX
int maxSensorValue = 0;   // Globale Integer Variable maxSensorValue
```

Selbst geschrieben Funktionen können nach der loop Funktion implementiert werden.

```
void TestServo()
{
	if (maxSensorValue > 10)
	{
		yourServoX.write(45);
	}
	else
	{
		yourServoX.write(135);
	}
}
```

3. Hello World Programm und Serial Monitor

Der Serial Monitor ermöglicht es vom Arduino gesendeten Text in einem Fenster am PC auszugeben.

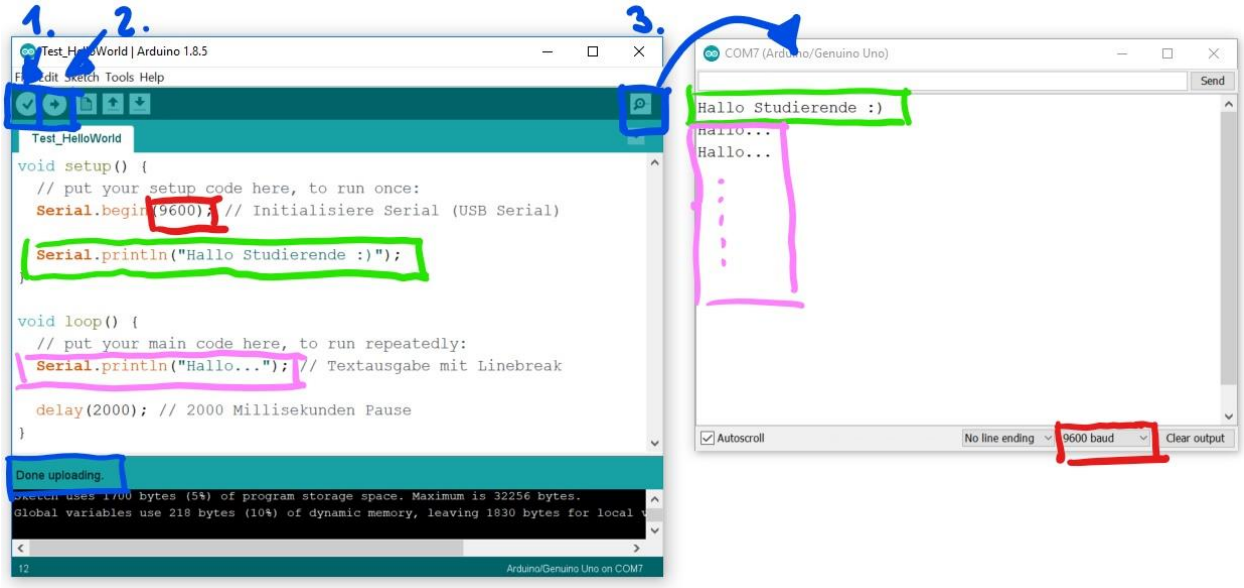
Mit den Befehlen Serial.print(...); Serial.write(...); können Text oder Zahlenwerte als Text gesendet werden.
Die Befehle Serial.println(...); Serial.writeln(...); senden einen Zeilenumbruch nach dem Text.

Die Übertragungsgeschwindigkeit (Baudrate) müssen bei Arduino und dem Serial Monitor gleich sein.

Im Beispiel Programm ist die Serielleuebertragungsgeschwindigkeit mit 9600 Baud (Bits pro Sekunde) gewählt.

Beispiel Programm:

```
void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600); // Initialisiere Serial (USB Serial)

    Serial.println("Hallo Studierende :)");
}

void loop() {
    // put your main code here, to run repeatedly:
    Serial.println("Hallo..."); // Textausgabe mit Linebreak

    delay(2000); // 2000 Millisekunden Pause
}

// 1. Kompilieren sie den SourceCode
// 2. Übertragen sie das Programm auf den Arduino
// 3. Öffnen sie den Serial Monitor
```



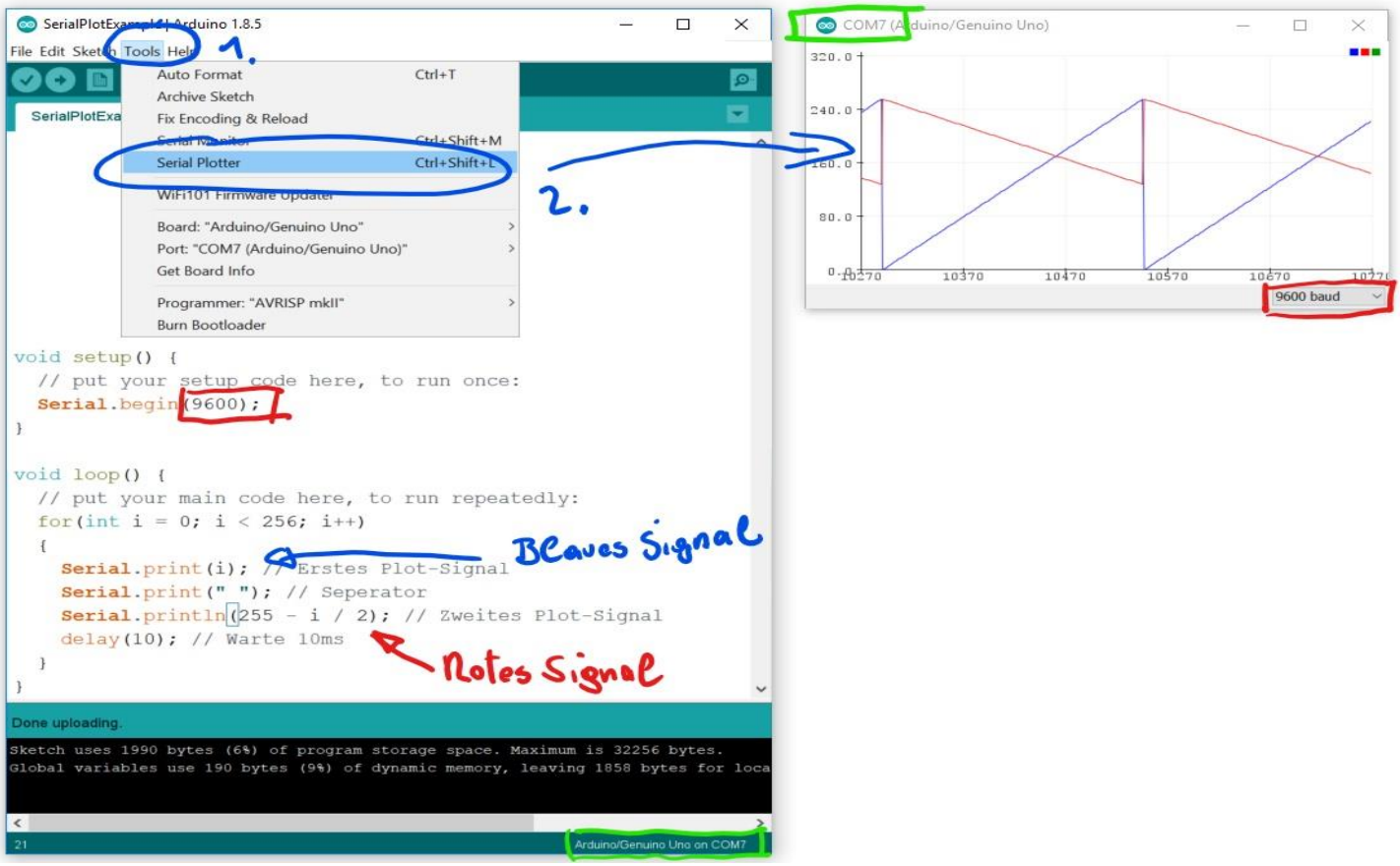
Wenn sie die Text-Ausgabe im Serial Monitor sehen, **wissen** sie dass ihre Einstellungen richtig sind, der SourceCode kompiliert und übertragen wurden und das der Arduino funktioniert.

3.1 Serial Plotter

Der Serial Plotter ermöglicht es Signale und Werte in einem Plot einfach zu visualisieren. Hierfür müssen Werte mit einem Leer-Zeichen „ “ oder Tabulator-Zeichen „\t“ getrennt zeilenweise mit der Serial.print() oder Serial.println() Funktion gesendet werden.

Die Uebertragungsgeschwindigkeit (Baudrate) müssen bei Arduino und dem Serial Plotter gleich sein.

Im Beispiel Programm ist die Serielleuebertragungsgeschwindigkeit mit 9600 Baud (Bits pro Sekunde) gewählt.



Beispiel Programm:

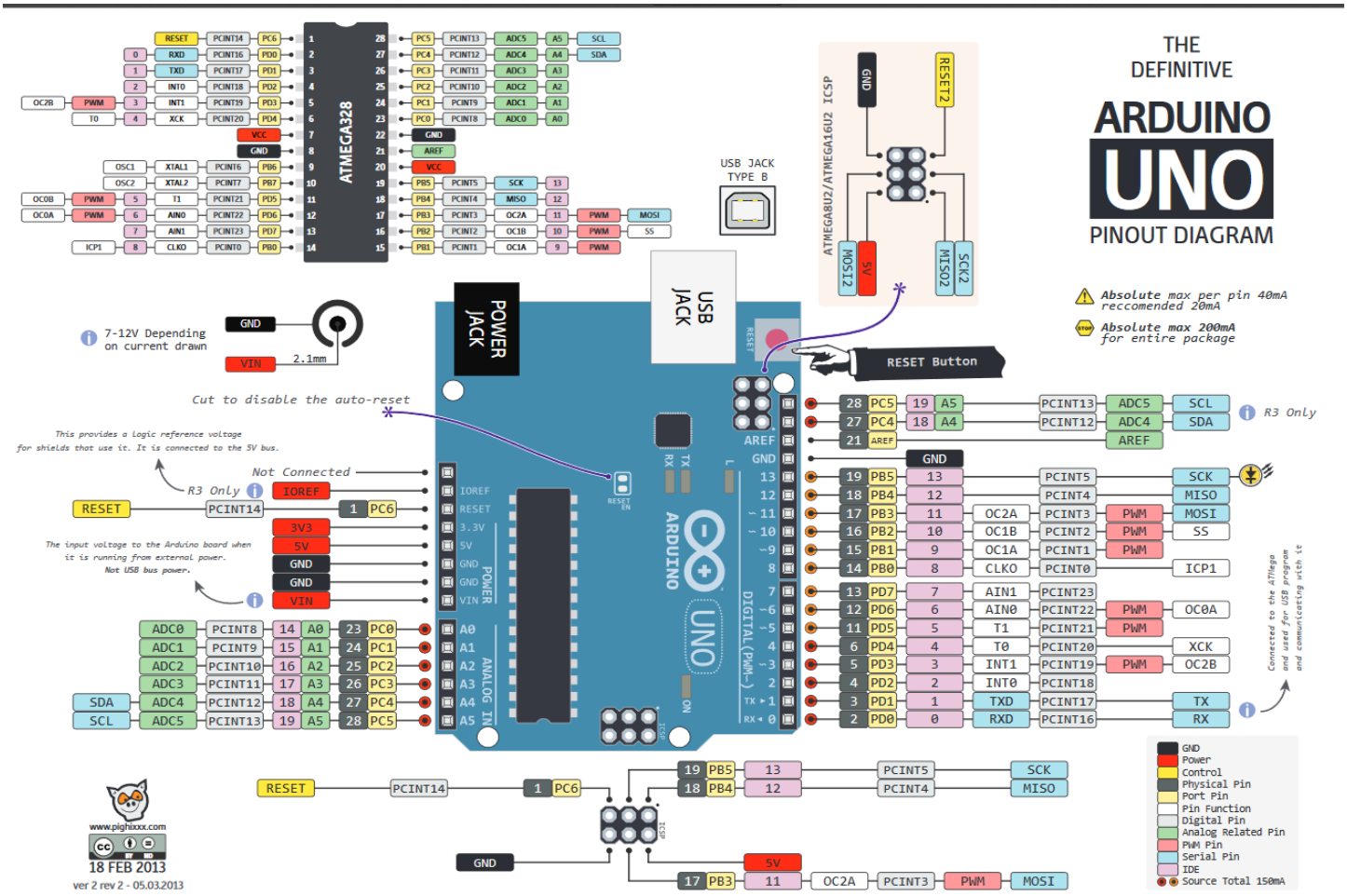
```
void setup() {

	// put your setup code here, to run once:
	Serial.begin(9600); // Baud-rate 9600 Baud
}

void loop() {
	// put your main code here, to run repeatedly:
	for (int i = 0; i < 256; i++) // Schleife mit Variable i von 0 bis 255
	{
		Serial.print(i); // Erstes Plot-Signal
		Serial.print(" "); // Separator
		Serial.println(255 - i / 2); // Zweites Plot-Signal und Zeilenumbruch
		delay(10); // Warte 10ms
	}
}
```

Der Serial Potter eignet sich sehr um gelesene Sensor-Werte auszugeben und ein Gefühl seiner Signal-Charakteristik zubekommen.

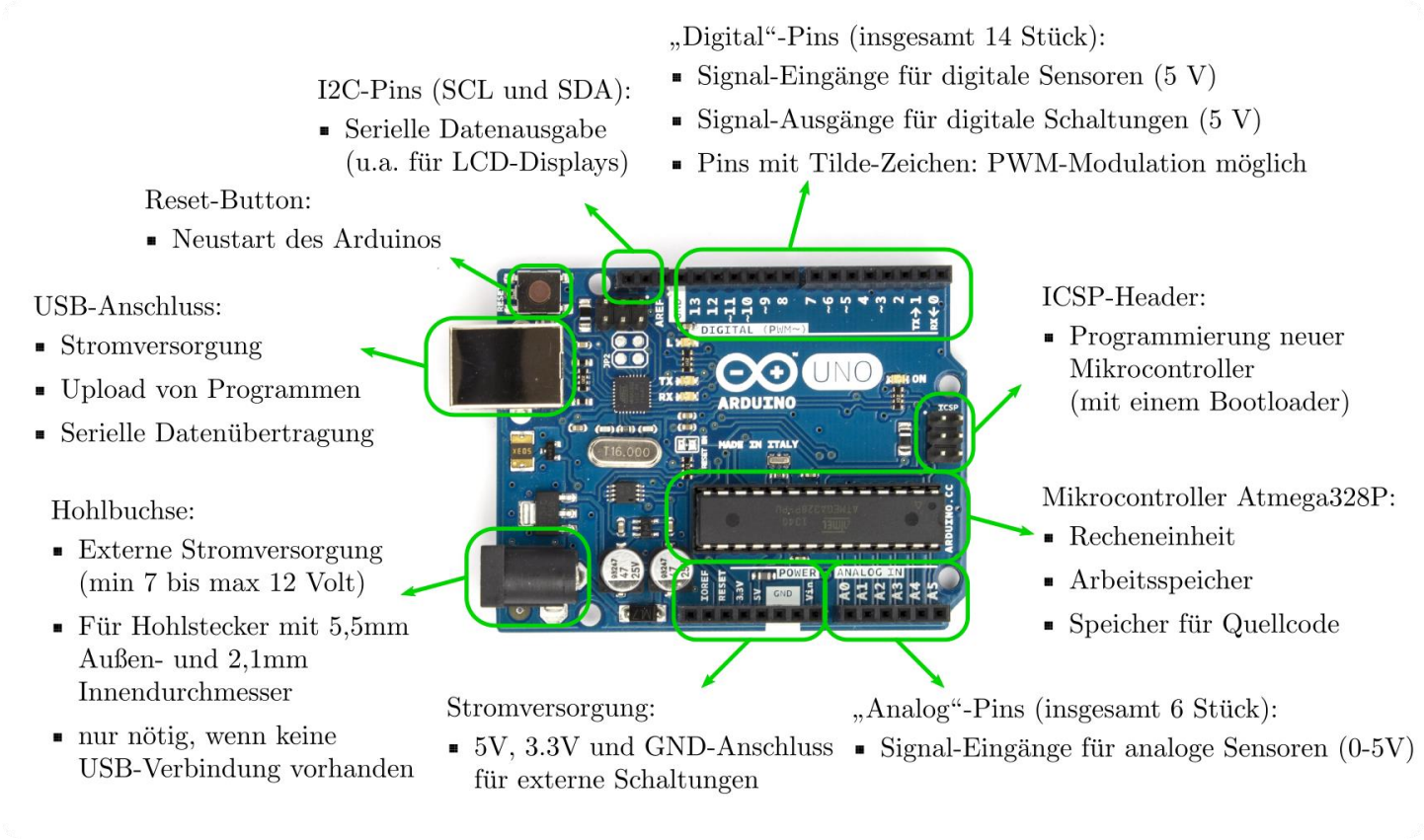
4. Was kann unser Board

Features	Arduino Uno	Arduino Mega (1280)	Arduino Mega 2560
Microcontroller (MCU)	ATmega328P (Datasheet)	ATmega1280	ATmega2560 (Datasheet)
Operating Voltage of the Microcontroller	5 V	5 V	5 V
Typical Supply Voltage for the board	7 V – 12V	7 V – 12 V	7 V – 12 V
Digital I/O Pins	**14 (includes 6 PWM outputs)**	**54 (includes 15 PWM outputs)**	**54 (includes 15 PWM outputs)**
PWM outputs	**6**	**15**	**15**
Analogue Input Pins	**6**	**16**	**16**
Digital Interface	**UART, I2C, SPI**	**UART, I2C, SPI**	**UART, I2C, SPI**
Max DC Current per I/O Pin	20 mA	20 mA	20 mA
Max DC Current for 3.3V Pin	50 mA	50 mA	50 mA
Flash memory of MCU	**32 KB**	**128 KB**	**256 KB**
Bootloader footprint	**0.5 KB**	**4 KB**	**8 KB**
SRAM of MCU	**2 KB**	**8 KB**	**8 KB**
EEPROM of MCU	**1 KB**	**4 KB**	**4 KB**
Clock Speed	16 MHz	16 MHz	16 MHz
Board Dimensions (Length)	**68.6 mm**	**101.5 mm**	**101.5 mm**
Board Dimensions (Width)	53.4 mm	53.4 mm	53.4 mm

http://www.elecrom.com/arduino-uno-vs-mega-2560/ https://www.arduino.cc/en/Products/Compare



http://marcusjenkins.com/wp-content/uploads/2014/06/ARDUINO_V2.pdf



https://www.grund-wissen.de/elektronik/arduino/aufbau.html

- Die digitalen Pins `0` bis `13` können als Sensor-Eingänge festgelegt werden:
  Eine anliegende Spannung von > `2.5V` (bzw. ~halbe Betriebsspannung) wird als `HIGH` (Zahlenwert `1`), eine niedrigere Spannung als `LOW` (Zahlenwert `0`) interpretiert.

- Die analogen Pins `A0` bis `A5` sind als Sensor-Eingänge zum Messen von Spannungswerten zwischen `0V` und `5V` geeignet; durch einen eingebauten 10Bit Analog-Digital-Wandler werden die gemessenen Spannungswerte auf einem Zahlenbereich von `0` (keine Spannung) bis `1023` (maximale Spannung, also `5V`) abgebildet.

  Bei anderen Boards wie den LilyPad oder den Teensy 3 ist die Betriebsspannung/Referenzspannung `3.3V` und somit entspricht ein Wert von `1023` der Spannung `3.3Volt`.

  Die Analoge-Referenzspannung kann bei den meisten Arduino Boards verändert werde.
  Mit dem Befehlt `analogReference(`DEFAULT | INTERNAL | EXTERNAL`)` kann die Referenzspannung festgelegt werden. https://www.arduino.cc/reference/en/language/functions/analog-io/analogreference/

  `analogReference(`EXTERNAL`);` legt die Referenzspannung auf die an Pin AREF anliegende Spannung fest.
  Z.B: Liegt an AREF eine Spannung von `1.5Volt` an so entspricht ein mit readAnalog gelesener Wert von `1023` der Spannung `1.5Volt`.

  Das verändern der Analogenreferenzspannung ermöglicht es Sensorsignale mit geringer Spannung mit höherer Genauigkeit auszulesen.
  Man sollte niemals eine höhere Spannung als die Betriebsspannung des Mikrokontrollers an einen PIN leiten!

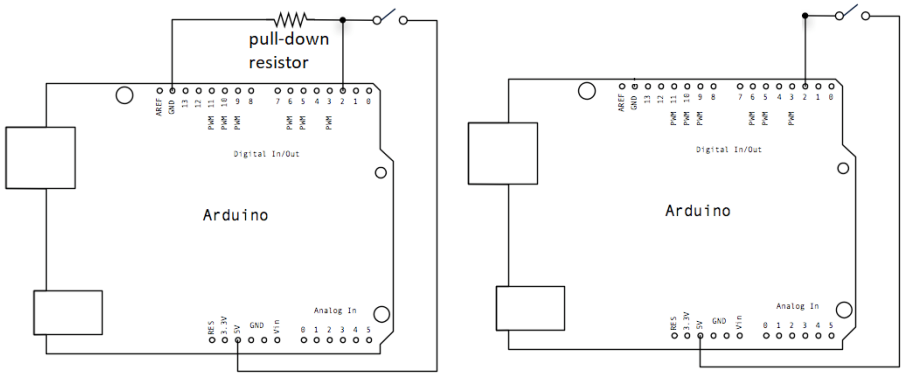
- PWM: Die mit dem Tilde-Zeichen ~ versehenen Pins (3, 5, 6, 9, 10, 11) können, wenn sie als Ausgabe-Pins festgelegt werden, zudem mittels einer so genannter Pulsweiten-Modulation (PWM) sehr schnell zwischen 0V und 5V hin und her wechseln. Man kann dabei Werte zwischen 0 und 255 angeben, wobei 0 für „immer aus“ und 255 für „immer an“ steht.

  Beispielsweise kann man mittels PWM einen Motor oder eine Glühbirne bei einem Wert von 128 mit nur „halber Leistung“ ansteuern, da er nur die Hälfte der Zeit mit Spannung versorgt wird und sich die andere Hälfte der Zeit im Leerlauf befindet.
  Eine LED lässt sich so ebenfalls „dimmen“: Die LED ist zwar schnell genug, um in der gleichen Frequenz mitzublinken, unser Auge jedoch nicht. Da wir nur 25 Einzelbilder je Sekunde wahrnehmen können, erscheint uns eine LED, die nur die Hälfte der Zeit an ist, gegenüber einer permanent hellen LED als dunkler.

- Wird ein Pin als Input konfiguriert aber nicht (elektrisch) verbunden so nennt man diesen einen floating Pin. Das elektrische potential bzw. der gelesene Wert ist undefiniert. Das bedeutet der Wert kann bei digitalRead (wild) zwischen 0 und 1 springen (bei analogRead auf Werte von 0 bis 1023).

  In der Abbildung unten sehen sie (rechts Without Pull-Down) das der Pin solange der Button/Taster nicht gedrückt ist floating ist.

  Damit bei einem Button/Taster der Wert nicht falsch gelesen wird muss dieser mit einem Widerstand (5k-1M Ohm, typisch ~10KOhm) auf ein definiertes elektrisches Potential „gezogen“ werden. Diesen Widerstand nennt man Pull-Up oder Pull-Down Widerstand.



With Pull-Down

Without Pull-Down

  Die meisten Mikrokontroller/Arduinos haben eingebaute Pull-Up Widerstände.
  Mit dem Befehl `pinMode(PinNr, INPUT_PULLUP);` kann dieser aktiviert werden.
  Ein Pin mit aktivierten internen Pull-Up ist somit im nicht verbundenem zustand auf HIGH (5V).

Arduino Programming Cheat Sheet

Primary source: Arduino Language Reference
http://arduino.cc/en/Reference/

Structure & Flow

Basic Program Structure

```
void setup() {
  // Runs once when sketch starts
}
void loop() {
  // Runs repeatedly
}
```

Control Structures

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
for (int i = 0; i < 10; i++) { ... }
break;     // Exit a loop immediately
continue;  // Go to next iteration
switch (var) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
return x;  // x must match return type
return;    // For void return type
```

Function Definitions

```
<ret. type> <name>(<params>) { ... }
e.g. int double(int x) {return x*2;}
```

Operators

General Operators

=	assignment		
+	add	-	subtract
*	multiply	/	divide
%	modulo		
==	equal to	!=	not equal to
<	less than	>	greater than
<=	less than or equal to		
>=	greater than or equal to		
&&	and	||	or
!	not		

Compound Operators

++	increment
--	decrement
+=	compound addition
-=	compound subtraction
*=	compound multiplication
/=	compound division
&=	compound bitwise and
|=	compound bitwise or

Bitwise Operators

&	bitwise and	|	bitwise or
^	bitwise xor	~	bitwise not
<<	shift left	>>	shift right

Pointer Access

&	reference: get a pointer
*	dereference: follow a pointer

Built-in Functions

Pin Input/Output

```
Digital I/O - pins 0-13 A0-A5
  pinMode(pin,
    [INPUT, OUTPUT, INPUT_PULLUP])
  int digitalread(pin)
  digitalWrite(pin, [HIGH, LOW])

Analog In - pins A0-A5
  int analogRead(pin)
  analogReference(
    [DEFAULT, INTERNAL, EXTERNAL])

PWM Out - pins 3 5 6 9 10 11
  analogWrite(pin, value)
```

Advanced I/O

```
tone(pin, freq_Hz)
tone(pin, freq_Hz, duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
  [MSBFIRST, LSBFIRST], value)
unsigned long pulseIn(pin,
  [HIGH, LOW])
```

Time

```
unsigned long millis()
  // Overflows at 50 days
unsigned long micros()
  // Overflows at 70 minutes
delay(msec)
delayMicroseconds(usec)
```

Math

```
min(x, y)   max(x, y)   abs(x)
sin(rad)    cos(rad)    tan(rad)
sqrt(x)     pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)
```

Random Numbers

```
randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)
```

Bits and Bytes

```
lowByte(x)    highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)  // bitn: 0=LSB 7=MSB
```

Type Conversions

```
char(val)     byte(val)
int(val)      word(val)
long(val)     float(val)
```

External Interrupts

```
attachInterrupt(interrupt, func,
  [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()
```

Variables, Arrays, and Data

Data Types

boolean	true | false
char	-128 - 127, 'a' '$' etc.
unsigned char	0 - 255
byte	0 - 255
int	-32768 - 32767
unsigned int	0 - 65535
word	0 - 65535
long	-2147483648 - 2147483647
unsigned long	0 - 4294967295
float	-3.4028e+38 - 3.4028e+38
double	currently same as float
void	i.e., no return value

Strings

```
char str1[8] =
  {'A','r','d','u','i','n','o','\0'};
  // Includes \0 null termination
char str2[8] =
  {'A','r','d','u','i','n','o'};
  // Compiler adds null termination
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numeric Constants

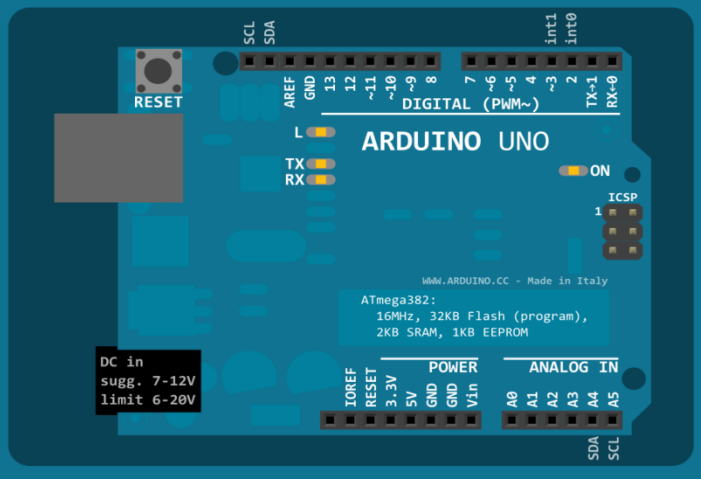
123	decimal
0b01111011	binary
0173	octal - base 8
0x7B	hexadecimal - base 16
123U	force unsigned
123L	force long
123UL	force unsigned long
123.0	force floating point
1.23e6	1.23*10^6 = 1230000

Qualifiers

static	persists between calls
volatile	in RAM (nice for ISR)
const	read-only
PROGMEM	in flash

Arrays

```
int myPins[] = {2, 4, 8, 3, 6};
int myInts[6];   // Array of 6 ints
myInts[0] = 42;  // Assigning first
                 // index of myInts
myInts[6] = 12;  // ERROR! Indexes
                 // are 0 though 5
```



Libraries

```
Serial - comm. with PC or via RX/TX
begin(long speed)  // Up to 115200
end()
int available()  // #bytes available
int read()   // -1 if none available
int peek()   // Read w/o removing
flush()
print(data)     println(data)
write(byte)     write(char * string)
write(byte * data, size)
SerialEvent()  // Called if data rdy

SoftwareSerial.h - comm. on any pin
SoftwareSerial(rxPin, txPin)
begin(long speed)  // Up to 115200
listen()       // Only 1 can listen
isListening()  // at a time.
read, peek, print, println, write
  // Equivalent to Serial library

EEPROM.h - access non-volatile memory
byte read(addr)
write(addr, byte)
EEPROM[index]  // Access as array

Servo.h - control servo motors
attach(pin, [min_uS, max_uS])
write(angle)  // 0 to 180
writeMicroseconds(uS)
   // 1000-2000; 1500 is midpoint
int read()    // 0 to 180
bool attached()
detach()

Wire.h - I²C communication
begin()     // Join a master
begin(addr) // Join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(byte)              // Step 2
send(char * string)
send(byte * data, size)
endTransmission()       // Step 3
int available() // #bytes available
byte receive()  // Get next byte
onReceive(handler)
onRequest(handler)
```


by Mark Liffiton

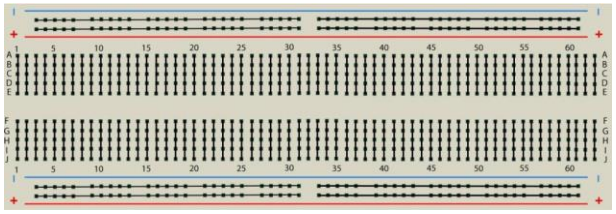
Adapted from:
- Original: Gavin Smith
- SVG version: Frederic Dufourg
- Arduino board drawing: Fritzing.org

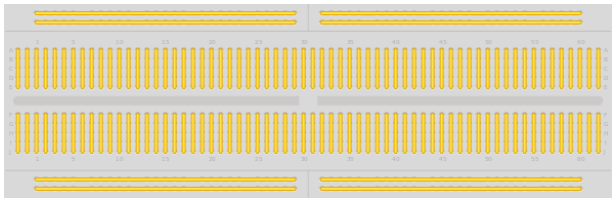
5. Steckbrett / Breadboard

Eine Steckplatine (englisch breadboard), auch „Steckbrett“, dient der mechanischen Befestigung und der elektrischen Verbindung von elektronischen Bauteilen für Versuchsschaltungen und Experimente.

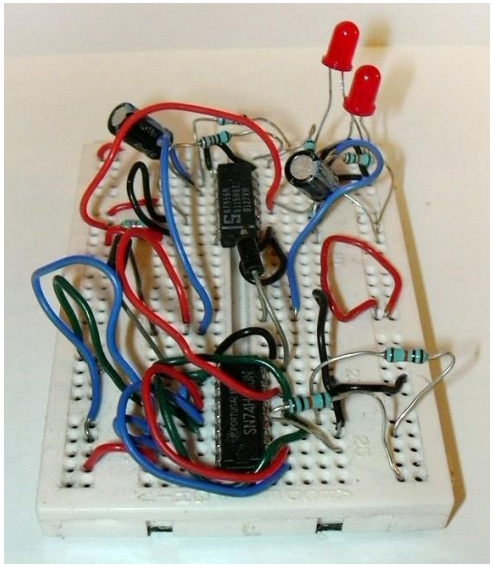
Im Gegensatz zu Leiterplatten werden bei Steckplatinen die Bauteile nicht gelötet, sondern in Federkontakte gesteckt. Dadurch kann die Schaltung durch einfaches Umstecken geändert werden. Steckplatinen werden häufig im Hobbybereich und teilweise auch in Schulen/Ausbildung verwendet, da der Aufbau schnell vonstattengeht und kein Löten erforderlich ist. In Elektronik-Experimentierkästen werden meistens ähnliche Stecksysteme benutzt.

Eine Alternative zur Steckplatine sind Lochrasterplatinen, auf denen elektrische Schaltungen flexibel aufgelötet werden können. [https://de.wikipedia.org/wiki/Steckplatine]





Die Gelben Linien zeigen wie die Steckloecher miteinander verbunden sind.



Beispiel eines Steckbrettes mit Schaltungsaufbau