Was ist die Syntax?

Die Syntax ist die Rechtschreibung. Ganz wichtig ist, dass (fast) jeder Befehl mit einem Semikolon (`;`) beendet wird. Bei Codeblöcken müssen immer geschweifte Klamern (`{}`) verwendet werden.

Das Grundgerüst

Bei jedem Arduino-Projekt muss man ein Grundgerüst "bauen". Das sieht dann so aus:

```
void setup() {
}

void loop() {
}
```

Void setup wird einmalig ausgeführt. Dort werden die Ein- und Ausgänge definiert und vieles andere. In Void loop kommt dann das rein, was das Programm machen soll, weil es unendlich lange ausgeführt wird. Zwischen die beiden Geschweiften Klammern wird dann der Code(Befehle) geschrieben.

Variablen

Danach müssen wir meistens Variablen definieren. Variablen sind so etwas wie Namensschilder, also können wir wichtige zahlen oder anderes z.B. Pin-nummern mit einem Namen versehen um sie besser wiederzuerkennen und der Code übersichtlicher blibt Am wichtigsten sind die int-Variablen und das geht so:

```
    int Variablenname = 13;
```

Hier haben wir eine Variable namens Variablenname definiert, die den Wert 13 hat. Den Namen kann man sich selber aussuchen.

Pins Anschalten und Ausschalten

Nun willst du sicherlich einen Pin anschalten oder aus. Das machst du mit digitalWrite. So schaltet man einen Pin an:

```
    digitalWrite(ledpin, HIGH);
```

Und so aus:

```
    digitalWrite(ledpin, LOW);
```

pausen

Wenn wir z.B eine Led blinken lassen wollen muss sie eine kurze pause machen bevor sie wieder an oder aus geht. Daführ nutzen wir die Funktion delay. So lassen wir etwas blinken:

```
    digitalWrite(led, HIGH);
    delay(pause);
    digitalWrite(led, LOW);
    delay(pause);
```

Hier brauchen wir zwei Variablen einmal für die led und einmal für pause, also die zeit, wo das programm anhält. In die klammer kann man auch einfach eine Zahl schreiben.

Operatoren

Im gleich folgenden Kapitel geht es um Bedingungen, also um Vergleiche. Dafür benötigen wir Operatoren.

- == beides soll gleich sein
- >= wenn die Variable größer oder gleich sein soll als der Vergleichswert
- <= wenn die Variable kleiner oder gleich sein soll als der Vergleichswert
- > wenn die Variable größer sein soll
- < wenn die Variable kleiner sein soll
- != wenn die Variable ungleich sein soll
- !> wenn die Variable ungleich oder größer sein soll
- !< wenn die Variable ungleich oder kleiner sein soll

Bedingungen

Wenn du möchtest, dass etwas nur dann ausgeführt wird kannst du die if-bedingungen benutzten:

```
    if (variable operator vergleichswert) {
        /* ... */
    }
```

wenn die Begingung erfüllt wird, wird der im Codeblock stehende Code ausgeführt, wenn nicht wird er übersprungen. Das könnte dann so aussehen:

```
if (schalter == HIGH) {
    digitalWrite(led, HIGH);
}
```

Wir können die Led auch ansonsten ausmachen wenn das nicht zutrifft und das geht mit else:

```
if (schalter == HIGH){
    digitalWrite(led, HIGH);
} else {
    digitalWrite(led, LOW);
}
```

Drück einen Knopf

Sicher möchtest du irgendwann, dass der Arduino etwas tut wenn du einen Knopf drückst. So kannst du das machen:

```
tasterstatus = digitalRead(taster);
if (tasterstatus == HIGH){
    /* ... */
}
```

So jetzt brauchen wir erstmal eine Variable mit dem Namen tasterstatus und dem Namen taster, was der Pin ist. Du kannst das allerdings auch direkt in eine if-bedingung schreiben:

```
if (digitalRead(taster) == HIGH){
    /* ... */
}
```

Analog auslesen

So jetzt haben wir schon mal einen Schalter oder taster ausgelesen, aber man kann da doch auch Sensoren Auslesen. Wer hätte es gedacht wir benutzen daführ analogRead. Hierbei haben wir wieder zwei möglichkeiten. Diesmal jedoch werden werte zwischen 0 und 1023 ausgegeben

```
sensordaten = analogRead(sensorpin);
```

Schleifen

Sicher willst du irgendwann, dass etwas immer wieder wiederholt wird wärend die bedingung erfüllt ist. dafür gibt es while.

```
    while (taster == HIGH){
        /* ... */
    }
```

Dann gibt es noch die for-Schleife. Mit ihr können wir einen Vorgang belibig oft wiederholen ohne, dass wir es mermals schreiben müssen:

```
    for i in range(0,5) {
        /* ... */
    }
```

Diese Schleife wird nun 6 mal wiederholt, denn ein Computer beginnt immer mit 0 beim zählen.

Funktionen

Nun möchtest du zum beispiel mehrere Befehle nacheinander öfters ausführen. Daführ verwendest du Funktionen. Mit der folgenden Funktion lassen wir einen Ausgang 2 sec. an und 2 sec. aus gehen.

```
void blinken(pin) {
    digitalWrite(pin, HIGH);
    delay(2000);
    digitalWrite(pin, LOW);
    delay(2000);
}
```

In den Klammern steht die für nur diese Funktion geltende Variable Pin. Aufrufen können wir diese Funkion nun mit:

```
    blinken(13);
```

Nun blinkt die interne LED auf Pin 13.

Wollen wir nun auch z.B. eine Funktion schreiben, die x und y multipliziert machen wir das so.

```
int x = 2;
int y = 3;
int ergebnis;
```

```
int multplizieren(int x, int y) {
    ergebnis = x * y;
    return (ergebnis);
}
```

Wie du merkst wird nun, da wir einen Rückgabewert haben (ergebnis) statt dem void ein Variablentyp hingeschrieben. Diese Funktion rufen wir nun so auf:

```
    washabenwirnunraus = multiplizieren(y, y);
```

Nun hat die Variable washabenwirnunraus den Wert 2*3 also 6 Was in den Klammern für Variablen stehen ist egal(in unserem fall müssen es zahlen sein). Das x unf y in der Funktionsklammer steht für die interne Variablen. Deshalb können wir die Funktion auch so aufrufen:

```
    washabenwirnunraus = multiplizieren(4, 6);
```

In diesem Fall haben wir die Werte mit denen gerechnet werden soll von ahnd und nicht per variable reingeschrieben. In diesem Beispiel muss 4*6 = 24 rauskommen.