Übersicht über die wichtigsten Befehle

Programmstruktur

Ein Programm in Arduino besteht aus mindestens zwei Methoden. Eine Methode ist ein Anweisungsblock, also eine Gruppe von Befehlen, die durch den Aufruf des Methodennamens ausgeführt werden. Die Methode void setup() wird einmal ausgeführt, die Methode void loop() wird ständig wiederholt.

```
void setup(){

}

void loop(){

}
```

Variablen

Eine Variable ist ein Container für Werte des Typs der Variable. Variablentypen sind:

Variablentyp	Bedeutung	Beschreibung
int	Integer	ganze Zahlen (-32.768 bis 32.767)
long	ganze Zahlen	(-2 Milliarden bis 2 Milliarden)
float	Fließkommazahl	gebrochene Zahlen
char	Character	Alphanumerische Zeichen (Buchstaben, Zahlen, Sonderzeichen)
array	Variablenfeld	mehrere Werte eines Variablentyps können gespeichert werden

Variablentyp	Deklaration	Erläuterung
int	int meinWert = 10;	In diesem Beispiel wird der ganzzahligen (int) Variable meinWert der Wert 10 zugewiesen. Überall, wo man nun auf meinWert zugreift, erhält man 10.
long	long meinWert = 1000;	Reicht der Wertebereich von int nicht aus, so kann man auf den Variablentyp long zurückgreifen. Ganze Zahlen von -2 Milliarden bis 2 Milliarden können gespeichert werden.
float	float meinWert = 2.5;	Oft benötigt man gebrochene Zahlen. Diese Werte können mit dem Variablentyp float gespeichert werden:
char	char meinBuchstabe = ‘a‘;	Um z.B. einen Buchstaben zu speichern benötigt man den Variablentyp char. Werte werden in einfachen Anführungszeichen (Minutenstrich) übergeben.
array	int meineWerte[5] = {10,12,32,46,50};	Bei Arrays handelt es sich im Grunde nicht um einen eigenen Variablentyp, sondern um eine Gruppierung mehrerer Variablen eines Typs. Im Beispiel wird als erstes ein Array vom Typ int angelegt. Die 5 in eckigen Klammern hinter dem Variablennamen bestimmen die Anzahl der Speicherplätze, die das Array bereit stellt. Man nennt die Anzahl der Speicherplätze auch die Länge des Arrays.

Befehle

Befehle sind Anweisungen, die Methoden in der Arduino-Software aufrufen.

Befehl	Deklaration	Erläuterung
pinMode()	pinMode(3,OUTPUT); // setzt den digitalen Kanal 3 als Ausgang	Der Befehl pinMode(Pin,Modus) deklariert einen digitalen Kanal auf dem Arduino-Board entweder als Eingang (INPUT) oder Ausgang (OUTPUT). Er bekommt als zusätzliche Informationen den Pin (Kanal) und die Funktion.
digitalWrite()	digitalWrite(3,HIGH); // Schaltet 5V+ auf den digitalen Kanal 3	Der Befehl digitalWrite(Pin,Wert) schaltet einen, zuvor mit pinMode() als OUTPUT deklarierten, digitalen Kanal auf HIGH (5V+) oder LOW (GND).
digitalRead()	digitalRead(4); // liefert HIGH oder LOW	Der Befehl digitalRead(Pin) liest ein digitales Signal am Übergebenen digitalen Kanal aus. Der Kanal muss vorher als INPUT deklariert worden sein.
analogWrite()	analogWrite(3,200); // am digitalen Kanal 3 wird werden 4V+ angelegt	Der Befehl analogWrite(Pin, Wert) setzt eine Spannung auf einen PWM-Kanal (digitale Kanäle mit PWM gekennzeichnet: 3, 5, 6, 9, 10, 11). Die Spannung wird als Wert zwischen 0 (GND) und 255 (5V+) übergeben.
analogRead()	analogRead(1); // liefert den anliegenden Wert vom analogen Kanal 1	Der Befehl analogRead(Pin) liest das anliegende Signal am übergebenen analogen Input ein. Digitale Kanäle können dafür nicht verwendet werden.
delay()	delay(1000); // der Programmablauf wird eine Sekunde verzögert	Der Befehl delay(Wert) verzögert den Porgrammablauf um den Wert in Millisekunden.
Serial.begin()	void setup(){ Serial.begin(9600); // Startet die Datenübertragung mit 9600 Baud}	Der Befehl Serial.begin(Baudrate) startet die Serielle Kommunikation zwischen dem Arduino-Board und dem Computer. Auslesen kann man die Daten z.B. im seriellen Monitor der Arduino-Software. Die möglichen Baudraten sind standardisiert: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, und 115200. Der Befehl muss im void setup() ausgeführt werden.
Serial.println()	Serial.println(analogRead(1)); // Sendet den analogen Wert am Kanal 1 an den Computer	Der Befehl Serial.println(Daten) sendet Daten über die serielle Schnittstelle. Man kann sie mit dem seriellen Monitor anzeigen lassen.

Methoden

Methoden sind Programmanweisungsblöcke. Wiederkehrende Abfolgen von Befehlen können in Methoden sinnvoll strukturiert werden. Parameter können an Methoden übergeben und Werte zurückgeliefert werden.

Beispiel	Deklaration	Erläuterung

Eine einfache Methode könnte so aussehen:	`void blinken(){`
	`// Anweisungsblock Start`
	`digitalWrite(ledPin, HIGH);`
	`delay(500);`
	`digitalWrite(ledPin, LOW);`
	`delay(500);`
	`// Anweisungsblock Ende`
	`}`
		Nun kann man die Methode z.B. aus dem void loop() aufrufen mit blinken();.

Parameter lassen sich auch an Methoden übergeben. Die Struktur sieht so aus:	`void blinken(int thePin, int dauer){`
	`digitalWrite(thePin, HIGH);`
	`delay(500);`
	`digitalWrite(thePin, LOW);`
	`delay(500);`
	`}`
		Hierbei wird der Parameter thePin und dauer übergeben. Der Aufruf kann dann so erfolgen: blinken(3,1000);.

Man kann auch einen Wert von der Methode zurück geben lassen. Dafür verwendet man anstelle von void den Variablentyp, den das Ergebnis haben wird und liefert es am Ende des Anweisungsblockes mit dem Schlüsselwort return an die Methode.	`float quadrat(float x){`
	`float ergebnis = x*x;`
	`return ergebnis;`
	`}`
		Der Aufruf wäre z.B.:
		`wert = quadrat(12.3);`

Operatoren

Operatoren sind mathematische oder logische Funktionen. Hier die wichtigsten im Überblick.

Arithmetische Operatoren

Operator	Bedeutung	Anwendung	Funktion
=	Zuweisung	a=2*b	Weist der linken Seite den Wert auf der Rechten Seite zu.
+	Addition	a=b+c	
-	Subtraktion	a=b-c	
++	Inkrementieren	a++	Zählt zur der Variable 1 hinzu (+1)
–	Dekrementieren	a–	Zieht von der Variable 1 ab (-1)
*	Multiplikation	a=b*c	
/	Division	a=b/c	Dabei darf c nie gleich Null sein
%	Modulo	a=b%c a=7%5 ; a=2 a=10%5 ; a=0	Liefert den Rest bei der Division von b/c. Ist b durch c teilbar, so ist das Ergebnis = 0.

Vergleichsoperatoren

Operator	Bedeutung	Anwendung	Funktion
==	Gleichheit	a==b	Prüft auf Gleichheit.
!=	Ungleichheit	a!=b	Prüft auf Ungleichheit
<	kleiner als	a<b	
>	größer als	a>b	
<=	kleiner gleich	a<=b	
>=	größer gleich	a>=b	
Boolsche Operatoren (Können Wahr oder Falsch sein.)			
&&	UND	(a=2)&&(b=5)	Wen beide Seiten wahr sind, ist das Ergebnis auch wahr.
!!	ODER	(a=2)!!(b=5)	Wen eine oder beide Seiten wahr sind, ist das Ergebnis wahr.
!	NICHT	!(a=3)	Ist wahr, wenn a nicht 3 ist.

Abfragen

Eine Abfrage prüft, ob z.B. eine Variable einen bestimmten Wert hat. Abfragen können also den Programmablauf steuern.

if - Abfrage

Die if-Abfrage prüft, ob die Übergebene Bedingung wahr ist. Wenn sie wahr ist, wird der Anweisungsblock durchlaufen, ist sie falsch, kann man einen alternativen Anweisungsblock (else) ausführen lassen.

```
if (digitalRead(btnPin)==HIGH) {
// Anweisungsblock für wahr
digitalWrite(ledPin,HIGH);
} else {
// Anweisungsblock für falsch
digitalWrite(ledPin,LOW);

}
```

Wenn der Button am btnPin gleich HIGH ist, wird die LED am ledPin eingeschaltet (HIGH), sonst wird sie abgeschaltet (LOW).

switch – case - Abfrage

Will man einen Wert auf verschiedene Zustände prüfen, bietet sich die switch-case-Abfrage an. Die Struktur sieht so aus:

```
switch (meineVariable) {
case 1:
befehl1;
break;
case 2:
befehl2;
break;
default:
befehl3;
break;
}
```

Hierbei wird der Block case 1 ausgeführt, wenn meineVariable == 1 ist. Ist sie 2, wird der Block case 2 ausgeführt. Ist sie weder 1 noch 2, wird der Block default ausgeführt. Wichtig sind die break-Befehle. Sie veranlassen das Programm, aus der Abfrage zu springen.

Schleifen		
Schleifen können Anweisungen bis zum Erreichen einer Abbruchbedingung wiederholen.		
for - Schleife		

Die for-Schleife hat folgende Struktur:	for (int i=0; i<10; i++){
	// Anweisungen
	}
		Als Parameter werden in den Klammern die Initialisierung (int i=0), die Abbruchbedingung(i<10) und die Fortsetzung (i++) übergeben.
		Alle Anweisungen, die in den geschweiften Klammern stehen, werden solange wiederholt, bis die Abbruchbedingung erfüllt ist.

do – while - Schleife		

Die do-while Schleife wird auch so lange wiederholt, bis eine Abbruchbedingung eintrifft, allerdings sorgt die do-while Schleife nicht selbst dafür.	do
	{
	delay(50);
	x = analogRead(3); // prüft den Sensorwert am Pin 3
	} while (x < 100);
		-