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.

← Operatoren

Schleifen →

Befehle

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

pinMode()

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.

```
pinMode(3,OUTPUT);  // setzt den digitalen Kanal 3 als Ausgang&lt;strong&gt;&lt;
```

digitalWrite()

Der Befehl digitalWrite(Pin,Wert) schaltet einen, zuvor mit pinMode() als OUTPUT deklarierten, digitalen Kanal auf HIGH (5V+) oder LOW (GND).

```
digitalWrite(3,HIGH); // Schaltet 5V+ auf den digitalen Kanal 3
```

digitalRead()

Der Befehl digitalRead(Pin) liest ein digitales Signal am Übergebenen digitalen Kanal aus. Der Kanal muss vorher als INPUT deklariert worden sein.

```
digitalRead(4); // liefert HIGH oder LOW
```

analogWrite()

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.

```
analogWrite(3,200); // am digitalen Kanal 3 wird werden 4V+ angelegt
```

analogRead()

Der Befehl analogRead(Pin) liest das anliegende Signal am übergebenen analogen Input ein. Digitale Kanäle können dafür nicht verwendet werden.

```
analogRead(1); // liefert den anliegenden Wert vom analogen Kanal 1
```

delay()

Der Befehl delay(Wert) verzögert den Porgrammablauf um den Wert in Millisekunden.

```
delay(1000); // der Programmablauf wird eine Sekunde verzögert&lt;strong&gt;&lt;
```

Serial.begin()

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.

```
void setup(){
Serial.begin(9600); // Startet die Datenübertragung mit 9600 Baud
}&lt;strong&gt;&lt;/strong&gt;
```

Serial.println()

Der Befehl Serial.println(Daten) sendet Daten über die serielle Schnittstelle. Man kann sie mit dem seriellen Monitor anzeigen lassen.

```
Serial.println(analogRead(1)); // Sendet den analogen Wert am Kanal 1 an den Com
```

Bibliotheken

Bibliotheken (Libraries) erweitern den Funktionsumfang der Arduino-Software um weitere Befehle. Es gibt Bibliotheken für Servos, erweiterte serielle Kommunikation und viele mehr. Will man sie verwenden, müssen sie in den Sketch eingefügt werden. Im Hauptmenü findet man unter Sketch den Befehl Import Library. Hier wählt man einfach die Bibliothek aus, die man verwenden will und im Sketch erscheint die Include-Zeile. Z.B.:

```
1 #include &lt;Servo.h&gt;
```

Man kann auch eine neue Library hinzufügen. Der entpackte Ordner der Library muss in den Libraries-Ordner im Sketchbook kopiert werden. Existiert dieser Ordner nicht, muss man ihn anlegen. Nach dem Programm-Neustart steht die Bibliothek zum Import bereit.

← Schleifen

Netzteil →

Methoden

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

Eine einfache Methode könnte so aussehen:

```
1 void blinken(){  // Anweisungsblock Start
2   digitalWrite(ledPin, HIGH);
3   delay(500);
4   digitalWrite(ledPin, LOW);
5   delay(500);
6   // Anweisungsblock Ende
7 }
```

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:

```
1 void blinken(int thePin, int dauer){
2   digitalWrite(thePin, HIGH);
3   delay(500);
4   digitalWrite(thePin, LOW);
5   delay(500);
6 }
```

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.

```
1 float quadrat(float x){
2 float ergebnis = x*x;
3 return ergebnis;
4 }
```

Der Aufruf wäre z.B.:

```
1 wert = quadrat(12.3);
```

← Befehle

Operatoren →

Operatoren

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

Operator	Bedeutung	Anwendung	Funktion
Arithmetische Operatoren			
=	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			
==	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.

← Methoden

Abfragen →

Programmstruktur

Die grundlegende Programmstruktur eines Arduino-Programms setzt sich aus zwei Methodenblöcken zusammen. Die erste Methode ist void setup(). Hier werden Grundeinstellungen (z.B. ob ein Kanal ein In- oder Output ist) vorgenommen. Diese Methode wird nur beim Programmstart ausgeführt, also genau ein Mal.

Die *void loop()* Methode wird im Gegensatz zum Setup ständig wiederholt. Hier wird der eigentliche Programmablauf geschrieben.

Über dem Setup kann man noch Bibliotheken einbinden und globale Variablen deklarieren. (Wenn du nicht weißt, was global bedeutet, mach dir noch keinen Kopf darum. Das ist für die meisten Arduino-Programme völlig egal.)

```
1 // Bereich für das Einbinden von Bibliotheken und die Deklaration von Variablen.
2
3 void setup() {
4
5 }
6
7 void loop() {
8
9 }
```

Weiter mit Variablen.

Schleifen

Schleifen können Anweisungen bis zum Erreichen einer Abbruchbedingung wiederholen.

for-Schleife

Die for-Schleife hat folgende Struktur:

```
for (int i=0; i&lt;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 &lt; 100);
```

← Abfragen

Bibliotheken →

Variablen

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

Variablentyp	Bedeutung	Beschreibung
int	ganze Zahlen	ganze Zahlen (-32.768 bis 32.767)
long	ganze Zahlen	(-2 Milliarden bis 2 Milliarden) – gut, wenn man z.B. die abgelaufenen Millisekunden zählen will, da die schon mal über 32.767 gehen
float	Fließkommazahl	gebrochene Zahlen
char	Character	Alphanumerische Zeichen (Buchstaben, Zahlen, Sonderzeichen)
array	Variablenfeld	mehrere Werte eines Variablentyps können gespeichert werden

Variablentyp int

Der Variablentyp int speichert ganze Zahlen in einem Wertebereich von -32.768 bis 32.767.

```
1 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.

```
1 digitalWrite(meinWert, HIGH);
```

Variablentyp long

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.

```
1 long meinWert = 1000;
```

Variablentyp float

Oft benötigt man gebrochene Zahlen. Diese Werte können mit dem Variablentyp float gespeichert werden:

```
1 float meinWert = 2.5;
```

Variablentyp char

Um z.B. einen Buchstaben zu speichern benötigt man den Variablentyp char.

```
1 char meinBuchstabe = 'a';
```

Werte werden in einfachen Anführungszeichen (Minutenstrich) übergeben.

Arrays

Bei Arrays handelt es sich im Grunde nicht um einen eigenen Variablentyp, sondern um eine Gruppierung mehrerer Variablen eines Typs.

```
1 int meineWerte[5] = {10,12,32,46,50};
```

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.

Im Programm kann man durch indizierte Abfragen auf die Speicherplätze des Arrays zugreifen. Die erste Stelle im Array ist die Stelle 0: meineWerte[0], der Wert ist 10 usw.

```
1 analogWrite(ledPin, meineWerte[0]);
```

← Programmstruktur Befehle →