Zusammenfassung der Theorie, Teil 1

In der Reihenfolge des Unterrichtes

Serieller Monitor (SM)

Der SM wird im Setup durch `Serial.begin(9600);` initialisiert. Durch `Serial.print(„Text“);` wird Text ausgegeben und mit `Serial.print(n);` würde der aktuelle Wert der Variablen n angezeigt. Für eine neue Zeile lautet die Anweisung `Serial.println(„Text“);`

if-else Abfragen: `if (Bedingung) {mache irgendwas} else {mache etwas anderes}.` Je nach Problem ist der „else“-Teil nicht unbedingt notwendig.

Deklarieren, initialisieren: Eine **Variable** wird **deklariert** indem man ihr einen Namen zuweist und den Datentyp festgelegt.
`int Zeit;` „`int`“ gibt an, wieviel Speicherplatz „reserviert“ wird und „`Zeit`“ ist der Name der Variable. Datentypen und deren Mindestspeicherbedarf:

`bool`	1 Bit	`long`	4 Byte
`byte`	1 Byte	`float`	4 Byte
`int`	2 Byte		

`bool, byte, int` und `long` sind ganzzahlig (integer) und `float` steht für (Fließ)Kommazahlen.

Eine **Variable** sollte zusätzlich **initialisiert** werden, d.h. man gibt ihr einen Startwert. Beispiel:
`Pause=1000;` // Zuweisung oder Initialisierung
Zusammengefasst: `int Pause=1000;`
Es ist guter Programmstil,
* jeder Variable einen aussagekräftigen Namen zu geben,
* jede Variable bei der Deklaration zu initialisieren und
* mit einem Kommentar zu versehen, Beispiel:

```
int Pause=1000;   // Zeit in ms für die Dunkelpause zwischen
                     zwei Lichtimpulsen
```

* nie das Offensichtliche kommentieren, stattdessen einen verständlicheren Code schreiben

Info: Deklarationen, Initialisierungen und Anweisungen müssen **immer** mit Semikolon (;) abgeschlossen werden. Das was in den geschweiften Klammern steht heißt „Anweisungsblock“, danach darf kein Semikolon gesetzt werden

Die for-Schleife (Zählschleife)

Wie der Name schon sagt, benutzt man diese Hauptsächlich zum Ab - / Durchzählen von abzählbaren Mengen. Syntax:

```
for(int i=0; i<6; i++) {mach irgendwas;}
```

Allgemein: **for**(Zähler-Initialisierung; Bedingung; Zählerveränderung) **{**Anweisungen**}**.

Die while-Schleife

Bei dieser Schleife wird geprüft, ob die Bedingung zutrifft noch bevor die Schleife betreten wird. Ist dem nicht so, wird die Schleife übersprungen. Eine solche Schleife nennt man „kopfgesteuert“.
Die while-Schleife wird oft benutzt, wenn noch nicht absehbar ist, wie viele Durchläufe benötigt werden.

Syntax: `while(Bedingungen) {Anweisungen}`,
Beispiel:
`while(sensorLicht < 200){do something;}`.
Hier muss die Variable “sensorLicht” vorher deklariert werden.
Oder:

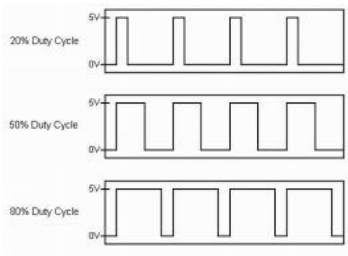
```
int n=0;   while(n<10) {mach etwas; n++;}
```

Ton erzeugen: Mit der Anweisung `tone(9, x, y);` erhältst du eine Tonausgabe, hier an Pin9, Frequenz x, Dauer y in ms. Die Dauer kann man auch weglassen. Dann kommt der Ton solange, bis er durch eine neue Ton-Anweisung oder `noTone(9);` überschrieben wird.

Info: **Subroutine**
bzw. Unterprogramm, Prozedur, evtl. Methode.
Subroutinen könnte man als Unterprogramme (Hilfsprogramme) verstehen. Subroutinen sind also keine lauffähigen Programme. Sie liefern nur Hilfsfunktionen, die von einem Programm aufgerufen werden.
void bedeutet „leer“, in einer solchen Subroutine wird nichts an das übergeordnete Programm zurückgegeben.

Info: Was sind **analoge Eingänge** und wie werden diese ausgewertet?
Viele Sensoren geben nicht `0` und `1` (also 0V oder 5V) aus, sondern auch Zwischenwerte. Dazu verfügt unser Arduino über sechs Stück **10 Bit Analogwandler** (*kurz:* A/D-Wandler) `A0` bis `A5`. Dies bedeutet, dass eine Eingangsspannung zwischen 0V und 5V auf die (binären) Werte 0…..1023 umgesetzt wird. Diesen Wert können wir z.B. mit `analogRead(A0);` abrufen (auslesen). Ist der Wert z.B. 512, so liegt an `A0` eine Spannung von 2,5V an.

PWM (Puls-Weiten-Modulation)



Umsetzung von digitalen Werten von 0 bis 255 (also eine 8 Bit-Auflösung) auf quasi analoge Werte durch Ein- und Ausschalten der Spannung am Ausgang. Beispiel: Mit `analogWrite(8, 64);` wird der Pin8 25% der Zeit eingeschaltet, da 64/256=0,25=25% ist. Wie oft pro Sekunde umgeschaltet wird, gibt der Arduino vor.

Teil 2

Info: Was ist ein **Array**? [engl. *array* = Feld, Ansammlung]
Ein Array ist ein **Datentyp**, mit welchem man beliebig viele Werte abspeichern kann. Beispiel:

```
int ledPin[6] = {7, 8, 9, 10, 11, 12};
```

[6] die „6" gibt die Anzahl der Elemente des Arrays an, „7" ist das Nullte Element, „8" das erste, „9" das Zweite usw.

Am Datentyp „int" erkennt der Compiler, dass es sich um eine Deklaration handelt. Schreibt man hingegen nur `ledPin[3]` , so wird das 3. Element ausgelesen, also ist `ledPin[3]=10`, da die Zählrichtung von links mit „0" beginnt.
Mit der Angabe des Datentypes z.B. „int" ist die Zahl in der eckigen Klammer die Anzahl der Elemente des Arrays, ohne Datentyp gibt die Zahl die Nummer des Elementes an, welches gelesen werden soll.

Zufallszahl (integer) im Bereich von a bis b: `random(a, b);`

Vergleiche: == gleich, != ungleich,
>, <, <=, >= größer, kleiner, kleiner oder gleich, größer oder gleich

Boolesche Operatoren: && und, || oder, ! nicht

Shift-Register (meistens ein Speicher für ein Byte um Daten zwischenzuspeichern, dient auch als seriell-parallel-Wandler).
dataPin: Hier müssen die Daten anliegen bevor sie ins Register eingelesen werden. Einlesen: Der **shiftPin** (clockPin) muss von Low auf High gehen. Speichern im Latch: Am **storePin** muss einen Wechsel von Low auf High stattfinden.
Mit **shiftOut**(dataPin, clockPin, bitOrder, value) kann man ein ganzes Byte (value) ins Register bringen, die Reihenfolge (bitOrder) ist mit MSBFIRST bzw. LSBFIRST wählbar.
Durch **bitSet(x,n)** lassen sich einzelne Bits im Register setzen. x ist eine Bytevariable, n die Stelle im Byte, an welcher eine „1" gesetzt wird. Die Zählweise im Byte beginnt links mit „0". Analog lassen sich durch **bitClear(x,n)** einzelne Bits löschen.
Weitere Informationen hierzu: Siehe Extrablatt „74HC595 8 Bit Schieberegister mit Latch".

Fortsetzung folgt.....