# Workshops Technologies Sem01

## Les 1

Deze les heb ik helaas gemist. Er ging iets mis met de opname, en ik was niet live aanwezig.

## Les 2

Les 2 vond ik vrij saai. Volgens mij ging het vooral over het inbouwen van een *delay* bij een button.

We leerden dat het gebruik van `delay()` niet ideaal is, omdat het programma tijdens een delay niets anders kan doen — het wacht simpelweg. Daardoor stopt alle andere logica tijdelijk. Dit maakt het onhandig bij programma's die op meerdere signalen of sensoren tegelijk moeten reageren.

In de les werd `delay()` zelf niet echt besproken, maar er werd wel gebruikgemaakt van `millis()`.
Ik ben later zelf gaan uitzoeken wat het verschil tussen deze twee is.

### verschil tussen `delay()` en `millis()`

- **`delay()`** pauzeert het programma volledig voor het opgegeven aantal milliseconden. Het is eenvoudig te gebruiken, maar het blokkeert alle andere taken.
- **`millis()`** geeft het aantal milliseconden sinds het programma gestart is. Het programma blijft ondertussen doorlopen, waardoor je meerdere dingen tegelijk kunt laten gebeuren.

**Kort gezegd:**
`delay()` is handig voor simpele demo's, maar `millis()` is beter voor echte projecten, omdat het *niet-blokkerend* werkt. Daardoor kun je bijvoorbeeld knoppen uitlezen en lampjes schakelen zonder dat het programma "bevriest".

## Les 3

Vanaf deze les begon het voor mij te klikken. Ik snapte eindelijk beter wat het verschil is tussen `void setup()` en `void loop()`.

- **`void setup()`**: Hierin initialiseer je alles wat één keer moet gebeuren bij het opstarten van de Arduino, zoals het instellen van pinmodes of het starten van de seriële verbinding.
- **`void loop()`**: Dit is de code die continu herhaald wordt — hier gebeurt dus het daadwerkelijke gedrag van je programma.

We hebben een lampje laten branden en geleerd over de functie `millis()`, die de tijd (in milliseconden) geeft sinds de Arduino is opgestart.

Ik begrijp nu dat je, als je iets bijvoorbeeld pas na 2 seconden wil uitvoeren, rekent vanaf de huidige tijd:
`millis() + 2000`.

Zodra de actuele tijd dat punt bereikt, voer je de actie uit. Anders gezegd: `millis()` telt vooruit, dus als je geen tijd optelt, werk je “in het verleden”.

Verder behandelden we:

- Het instellen van pinnen als input of output met `pinMode()`.
- Het verschil tussen `INPUT_PULLUP` en een normale input (bij `INPUT_PULLUP` is de knop ‘LOW’ als hij ingedrukt is).
- Het toevoegen van libraries via Tools → Manage Libraries.
- Het uitlezen van analoge pinnen.

Tot slot nog een tip uit de les: Arduino’s zijn vaak goedkoper in China, maar let erop dat ze een compatibele chip hebben, bijvoorbeeld de ATmega86.

## verschil tussen `unsigned long` en `int`

Hierbij wist ik dat een `int` 16 bits is en een `long` 32 bits, maar ik was benieuwd wat `unsigned` *precies doet.

Een normale `int` kan negatieve en positieve waarden bevatten (–32.768 tot 32.767).
Een `unsigned int` gebruikt dezelfde 16 bits, maar alleen voor positieve waarden (0 tot 65.535).
Voor `unsigned long` geldt hetzelfde principe, maar dan met 32 bits — dus een bereik van 0 tot 4.294.967.295.

Omdat `millis()` steeds oploopt en nooit negatief wordt, gebruikt Arduino hier standaard een `unsigned long` voor.

### `millis()`

Dit betekent dus dat `millis()` naar ongeveer 49,7 dagen weer op 0 begint.

# Les 4 – State Diagrams

In deze les leerden we over state diagrams, een alternatief / additive manier om logica te ontwerpen in plaats van met een flowchart.

We maakten een diagram in draw.io en zagen dat een *state diagram* reageert op gebeurtenissen (*events*) in plaats van simpelweg één vaste volgorde te volgen, zoals bij flowcharts.

Een belangrijk inzicht was dat het gebruik van te veel booleans al snel tot verwarring leidt. Met states kun je in plaats daarvan duidelijk aangeven in welke toestand het programma zich bevindt.

We maakten een voorbeeld met drie toestanden:

1. `WAIT_ON_GREEN_LED`
2. `WAIT_ON_BTN_PRESSED`
3. `WAIT_ON_BTN_RELEASED`

In de code werd dit vertaald naar een `switch (currentState)`-structuur met bijbehorende `case`-blokken:

```
switch (currentState) { case WAIT_ON_GREEN_LED: if (digitalRead(BTN_PIN) == LOW) { currentState = WAIT_ON_BTN_RELEASED; } else if (millis() > greenOnTime) { digitalWrite(GREEN_LED_PIN, HIGH); currentState = WAIT_ON_BTN_PRESSED; } break; // enzovoort... }
```

## Flowchart vs State Diagram

Een flowchart beschrijft de logica binnen één functie of proces, een vaste volgorde van stappen.

Een state diagram daarentegen beschrijft de verschillende toestanden waarin een programma kan verkeren en hoe het ertussen schakelt.

Flowcharts zijn dus handig voor kleine, lineaire stukken code (zoals een berekening of controle),
terwijl state diagrams beter zijn voor grotere systemen met wisselend gedrag, zoals een verkeerslicht of game loop.

# Les 5 – Registers

In deze les leerden we dat je niet altijd afhankelijk hoeft te zijn van de standaard Arduino-functies zoals `digitalWrite()` of `digitalRead()`.
Je kunt ook direct communiceren met de registers van de microcontroller, wat veel sneller en preciezer is.

Dat is vooral nuttig als je snelheid nodig hebt, bijvoorbeeld bij real-time toepassingen of nauwkeurige timing.

## Overzicht van registers: `DDRB`, `PORTB`, `PINB`

Deze registers horen bij de hardware van de Arduino.

- **`DDRB`** bepaalt of de pinnen *input* of *output* zijn.
- **`PORTB`** schrijft waarden naar de pinnen (aan/uit).
- **`PINB`** leest de actuele waarde van de pinnen.

Kort gezegd:

> `DDRB` → richting instellen
> `PORTB` → schrijven
> `PINB` → lezen

Directe register-aansturing is minder gebruiksvriendelijk, maar veel efficiënter.
Daarom wordt het vaak gebruikt door ervaren programmeurs of bij projecten waar elke microseconde telt.

# Disclaimer – Gebruik van ChatGPT

Dit document is mede opgesteld met behulp van ChatGPT (model GPT-5).
De tool is gebruikt om mijn ruwe aantekeningen te structureren, grammatica te verbeteren en uitleg toe te voegen bij technische onderdelen die ik zelf heb onderzocht.
Alle inhoud heb ik handmatig gecontroleerd, herschreven en aangevuld om te zorgen dat het aansluit bij mijn eigen leerervaring.

**Gebruikte prompts (samenvatting):**

- “Kun je mijn antwoorden over lessen 2 t/m 5 samenvoegen tot een vloeiend verslag?”
- “Maak het leesbaar en professioneel, maar behoud mijn toon.”
- “Zet het geheel om naar Markdown zodat ik het kan kopiëren.”
- “Voeg een nette disclaimer toe waarin vermeld staat dat het met ChatGPT is gemaakt en welke prompts ik heb gebruikt”