TRÄFF 3 AV 5

Digital input

Modul 3: **Knapp, buzzer, logik**

NAMNGIVNA VÄRDEN

`const int` och `int`.

Ni har redan använt `const int` — för pin-nummer.

Nu behöver ni också `int` (utan `const`) — för värden som **ska kunna ändras under tiden programmet kör**: räknare, tillstånd, sensormätningar.

> TUMREGEL
> `const int` = sätts en gång, aldrig mer.
> `int` = kan ändras.
> `bool` = sant eller falskt.

```
1 const int knappPin = 9;     // ändras aldrig
2 int lastState = HIGH;       // kan ändras
3 bool larmPaslaget = false; // kan togglas
```

Full förklaring av datatyper, scope och operatorer i kompendiets Bilaga A.

DIGITAL INPUT

Input ≠ Output.

Hittills har Arduinon bara skickat ström ut (OUTPUT).
Nu läser den av ett tillstånd in (INPUT).

```
pinMode(knappPin, INPUT_PULLUP);
// digitalRead(knappPin):
//   LOW = tryckt, HIGH = släppt
```

VARFÖR PULLUP?

En pinne som bara "lyssnar" utan koppling till plus eller minus *flyter* — läsningen blir slumpmässig. `INPUT_PULLUP` låter Arduinon själv dra pinnen till HIGH. När du trycker dras den till GND (LOW). Därför: **tryckt = LOW**, släppt = HIGH.

FATTA BESLUT

`if` / `else`.

Läs ett värde → jämför → agera olika beroende på resultatet.

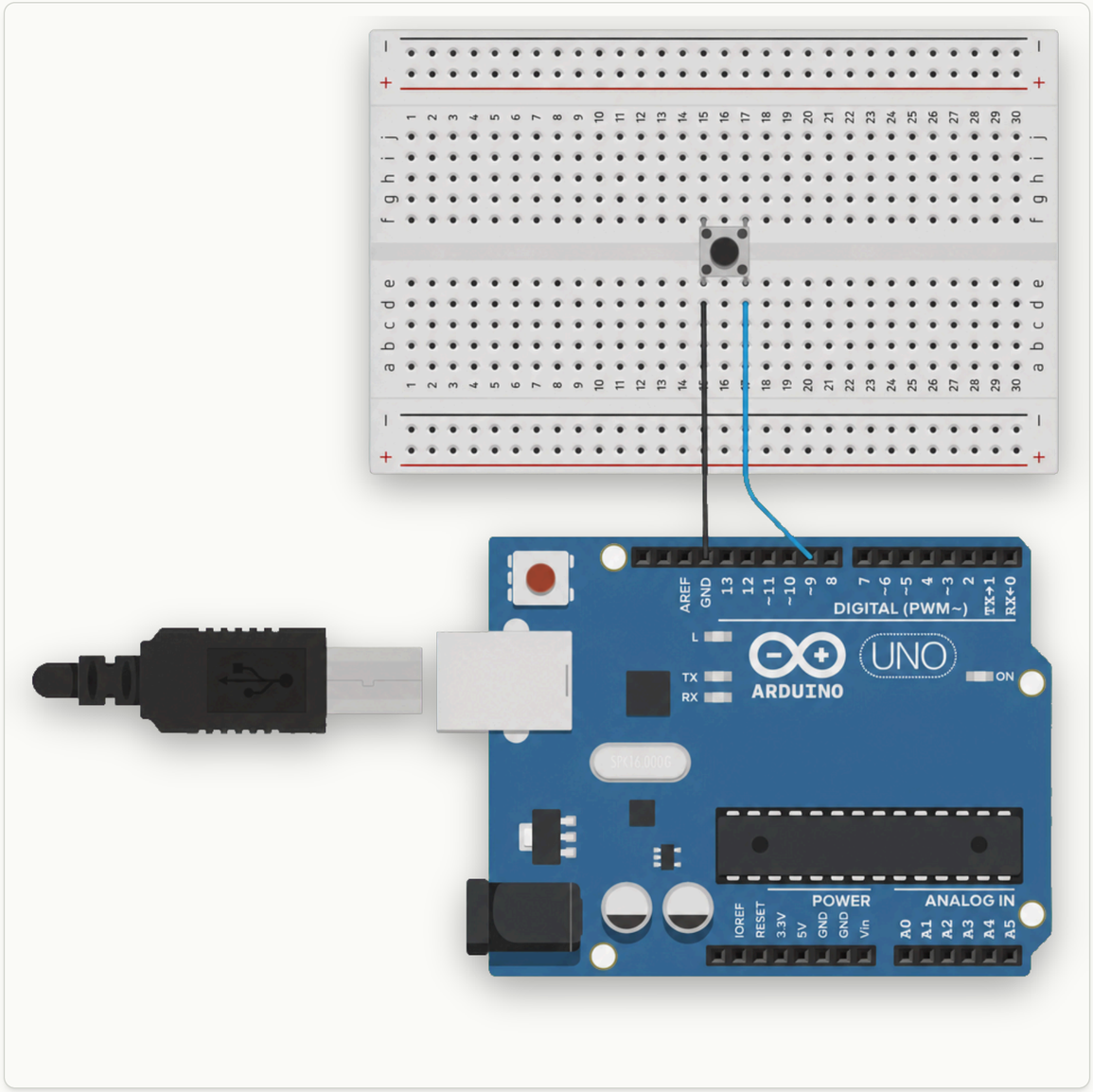
Operatorerna: `==` lika, `!=` olika, `<` mindre, `>` större.

> **VANLIGASTE NYBÖRJARFELET**
>
> = tilldelar värde.
> == jämför.
> Glöm inte andra likhetstecknet.

```
1  if (digitalRead(knappPin) == LOW) {
2    // knappen är tryckt
3    digitalWrite(LED_BUILTIN, HIGH);
4  } else {
5    // släppt
6    digitalWrite(LED_BUILTIN, LOW);
7  }
```

Fler operatorer (&&, ||, !)
och kedjade if/else i kompendiets Bilaga A.

Koppla knappen.



Vi använder bara `knapp A → D9` · andra benet till GND. Ingen pulldown-resistor — `INPUT_PULLUP` sköter det internt.

PEDAGOGISKT KRITISK

Reagera på flanken.

En knapp som **hålls nere** skulle toggla hundratals gånger per sekund. Resultat: larmet blinkar mellan av/på som ett stroboskop.

Lösningen: agera inte på att knappen *är* nere. Agera på att den **just nu gick från HIGH till LOW** — flanken.

> MÖNSTRET
> Spara förra värdet. Jämför med nuvarande. Om det ändrats åt rätt håll → det är en flank → agera en gång.

Inbyggda LED:en (pin 13) speglar `larmPaslaget` — så ni *ser* när larmet växlar.

```
1  bool larmPaslaget = false;
2  int lastState = HIGH;
3
4  void loop() {
5    int state = digitalRead(knappPin);
6    if (state == LOW && lastState == HIGH) {
7      larmPaslaget = !larmPaslaget;        // flank → toggla
8    }
9    digitalWrite(LED_BUILTIN, larmPaslaget ? HIGH : LOW);
10   lastState = state;
11   delay(10);  // mot studs
12 }
```

ACTIVE BUZZER

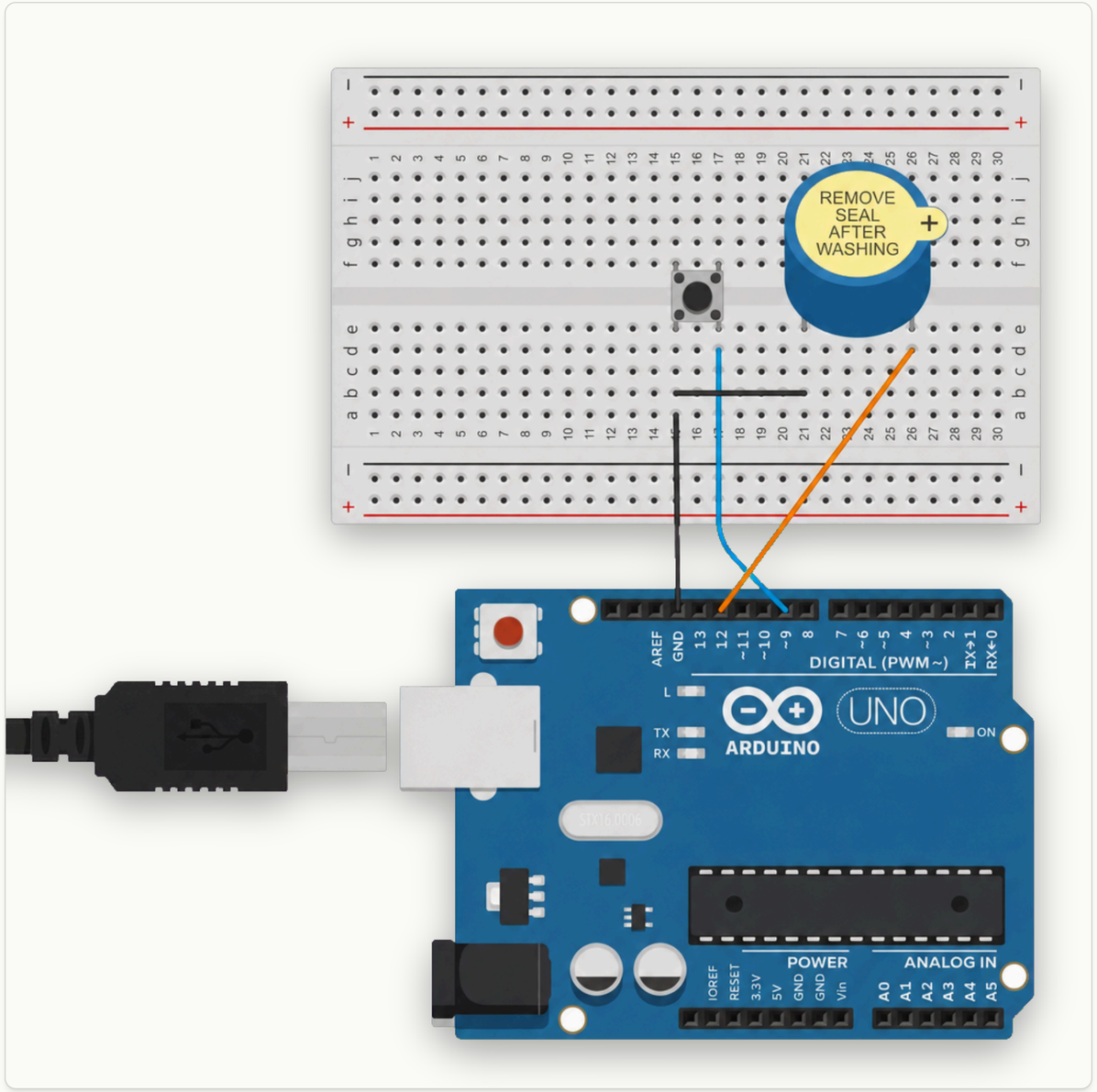
En varning först.

EXTREMT VIKTIGT

Dra absolut inte av den lilla klisterlappen `"REMOVE SEAL AFTER WASHING"` på er buzzer.

Utan lappen är buzzern obehagligt högljudd.
Lappen är vår ljuddämpare — den stannar på.

Koppla buzzern.



+-ben (höger) → D12 · −-ben → GND-skenan. Knappen står kvar — buzzern är tillägget. Ingen resistor.

SÄTT IHOP ALLT

Läs knapp. Styr buzzer.

Tryck → larm på (buzzern tjuter).
Tryck igen → tyst.

`INPUT → flank → OUTPUT`

SÄTT IHOP ALLT · LARM-MÖNSTRET

Kärnan i `loop()`.

```
int state = digitalRead(knappPin);
if (state == LOW && lastState == HIGH) {
  larmPaslaget = !larmPaslaget;           // flank → toggla
}
digitalWrite(buzzerPin, larmPaslaget ? HIGH : LOW);
lastState = state;
```

Samma mönster som "Reagera på flanken" — `LED_BUILTIN` bytt mot `buzzerPin`. Active buzzer: `HIGH` = pip, `LOW` = tyst.

EFTER TRÄFF 3

Arduinon lyssnar.

Ni har byggt Arduinons första sensor — en knapp — och gett den en röst — buzzern. Arduinon läser den ena — och styr den andra.

Nästa gång: analoga sensorer. Mörker, rörelse, mätvärden. Serial Monitor.