

# Syntax-grammatik

---

*Datatyper, operatorer, scope, funktioner — det lilla språket under Arduinos huva.*

## Vad "kod" egentligen är

En Arduino-sketch är ett program skrivet i C++. Det är ett fullfjädrat programmeringsspråk med egna regler för hur orden får kombineras — precis som svenska har grammatik och stavningsregler. Felar du mot dem vägrar kompilatorn att översätta din text till något Arduinon kan köra.

Kursens slides presenterar syntax genom exempel, men ingen formell översikt. Den här bilagan är den formella översikten. Den är inte uttömmande — den täcker **det ni använder på kursen**, inte hela C++.

## Variabler och datatyper

### DEKLARATION

En variabel är ett namngivet utrymme där du kan lagra ett värde. För att skapa en skriver du dess **datatyp**, dess **namn**, och (valfritt) ett **initialvärde**:

```
int count = 0;
bool larmPaslaget = false;
const int knappPin = 9;
```

- `int` är datatypen — heltal.
- `count` är namnet du hittat på.
- `= 0` är initialvärdet — ditt val.
- `;` avslutar raden. Glömmer du semikolonet klagar kompilatorn.

## PRIMITIVA DATATYPER SOM KURSEN ANVÄNDER

<code>int</code>	Heltal. -32768 till 32767 på Uno (16-bit). Använd när du räknar något: räknare, mätvärden, pin-nummer.
<code>long</code>	Stort heltal. Upp till ±2 miljarder. Använd <code>unsigned long</code> för <code>millis()</code> -tid eftersom den växer snabbt.
<code>byte</code>	Heltal 0–255. Perfekt för PWM-värden, bytes från sensorer, bitfält.
<code>bool</code>	Antingen <code>true</code> eller <code>false</code> . Använd för till/från-tillstånd.
<code>float</code>	Decimaltal. 3,14, -0,001 osv. Långsammare än <code>int</code> , undvik om du inte måste.
<code>char</code>	Ett enskilda tecken, typ <code>'a'</code> . Används sällan i denna kurs.

### CONST — "DET HÄR VÄRDET ÄNDRAS ALDRIG"

Prefixet `const` låser värdet efter det tilldelats. Försöker du skriva över det senare, vägrar kompilatorn.

```
const int ledPin = 13;    // bestämt nu, för evigt
int pressCount = 0;     // kan ändras
```

Använd `const int` för pin-nummer och andra fasta värden. Det gör koden läsbar och hindrar dig från att råka ändra värdet av misstag.

### CONST INT VS #DEFINE — SAMMA RESULTAT, OLIKA VÄGAR

I Arduino-exempel ser ni båda skrivsätten:

```
const int ledPin = 13;    // modern C++
#define LED_BUILTIN 13   // äldre C-makro
```

De fungerar ungefär likadant — båda ger namnet `ledPin` eller `LED_BUILTIN` värdet 13 — men under huven är de olika saker.

`const int` deklarerar en **riktig variabel** som lagras i minnet och som har en datatyp (`int`). Kompilatorn vet att det är ett heltal och kan stoppa dig från att använda det fel, till exempel skriva `ledPin = "text"`.

`#define` är ett **textmakro**: preprocessorn går igenom filen innan kompilatorn ser den, och ersätter varje förekomst av `LED_BUILTIN` med `13`. Ingen datatyp, ingen typkontroll. Det är ett äldre C-arv från 70-talet.

#### VARFÖR BÅDA?

Arduino-biblioteket (`Arduino.h`) använder `#define` av historiska skäl — makron fanns i C långt innan `const` blev standardiserad och typstark. För **er egen kod** på kursen använder vi `const int`, eftersom det är tydligare, typstarkt, och uppmärksammar dig

direkt om du råkar göra något galet. Att Arduino-biblioteket självt använder `#define` är helt OK — det är etablerat och välbeprövat.

## Operatörer

### ARITMETISKA

```
a + b    // addition
a - b    // subtraktion
a * b    // multiplikation
a / b    // division (heltal!)
a % b    // modulo – resten efter division
a++     // öka a med 1
a--     // minska a med 1
a += 5   // kortare form av: a = a + 5
```

### HELTALSDIVISION TRICKAR MÅNGA

`int a = 5 / 2;` ger `a = 2`, inte `2.5`. Heltal avrundas nedåt. För decimalsvar måste du använda `float`: `float a = 5.0 / 2.0;` ger `2.5`.

### JÄMFÖRELSE

```
a == b    // lika
a != b    // olika
a < b     // mindre än
a > b     // större än
a <= b   // mindre eller lika
a >= b   // större eller lika
```

**Det vanligaste nybörjarfelet:** att skriva `=` (tilldelning) istället för `==` (jämförelse).

```
if (x = 5) { ... } // FEL: tilldelar 5 till x, alltid sant
if (x == 5) { ... } // RÄTT: jämför x med 5
```

Kompilatorn varnar ibland, men inte alltid. Läs raden högt: ”om x är lika med 5” = `==`. ”Sätt x till 5” = `=`.

### LOGISKA

```
a && b    // OCH – båda måste vara sanna
a || b    // ELLER – minst en måste vara sann
!a        // INTE – vänder på sanningsvärdet
```

Används i `if`-villkor och tillstånd:

```
if (larmPaslaget && lutad) {  
  // både larm på OCH tilt lutad  
}  
  
if (!larmPaslaget || ljus > 500) {  
  // larm AV, ELLER det är ljus  
}
```

! framför en variabel är **logisk inversion** — det ”motsatta”. `!true = false`, `!false = true`. Används ofta för att togglar: `larmPaslaget = !larmPaslaget;`

## Villkorssatser

### IF / ELSE

```
if (villkor) {  
  // kör om villkor är sant  
} else {  
  // annars  
}
```

Flera alternativ kedjas med `else if`:

```
if (ljus < 200) {  
  // kolsvart  
} else if (ljus < 500) {  
  // halvmörkt  
} else {  
  // ljus  
}
```

Körs i ordning — första matchande gren vinner, resten hoppas över.

### KORT IF-SYNTAX (TERNÄR)

```
int värde = (villkor) ? omSant : omFalskt;  
  
// exempel:  
digitalWrite(LED_BUILTIN, larmPaslaget ? HIGH : LOW);
```

Läses: ”om larmPaslaget, sätt HIGH, annars sätt LOW”. En rad istället för fyra.

# Loopar

## FOR — UPPREPA ETT BESTÄMT ANTAL GÅNGER

```
for (int i = 0; i < 10; i++) {  
  // kör tio gånger, med i = 0, 1, 2, ..., 9  
}
```

Tre delar, åtskilda med semikolon:

1. **Initiering:** `int i = 0` — körs en gång innan loopen.
2. **Villkor:** `i < 10` — testas före varje varv. När det blir falskt, avslutas loopen.
3. **Uppdatering:** `i++` — körs efter varje varv.

Användbart för till exempel att tona en LED från släckt till tänd:

```
for (int v = 0; v <= 255; v++) {  
  analogWrite(ledR, v);  
  delay(10);  
}
```

## WHILE — UPPREPA TILLS NÅGOT ÄNDRAS

```
while (villkor) {  
  // körs så länge villkor är sant  
}
```

Kursens `void loop()` är i själva verket ett inbyggt `while`-loop som aldrig avslutas. Du skriver det inte själv, Arduino-ramverket gör det åt dig.

# Funktioner

En **funktion** är ett namngivet stycke kod du kan anropa. `setup` och `loop` är funktioner. Så är `digitalWrite` och `delay`. Du kan skriva egna också:

```
void blink(int tid) {  
  digitalWrite(LED_BUILTIN, HIGH);  
  delay(tid);  
  digitalWrite(LED_BUILTIN, LOW);  
  delay(tid);  
}
```

Delar:

- `void` är **returtypen** — vad funktionen ger tillbaka. `void` betyder ”inget”.
- `blink` är namnet.
- `(int tid)` är **parameterlistan** — värden som funktionen tar emot. Här ett heltal som heter `tid`.

- `{ ... }` är funktionens kropp — koden som körs.

Anropa den senare med `blink(200);` eller `blink(1000);`.

En funktion som **ger tillbaka** ett värde har annan returtyp:

```
int dubbelt(int x) {
  return x * 2;
}

// Användning:
int y = dubbelt(7); // y blir 14
```

## Scope — var en variabel syns

Variabler deklarerade **inne i** en funktion är **lokala** — de syns bara där.

```
void setup() {
  int temp = 5;
  // temp finns bara här inne
}

void loop() {
  // temp finns INTE här – kompilerings-fel om du försöker använda det
}
```

Variabler deklarerade **utanför** alla funktioner — längst upp i filen — är **globala** och syns överallt.

```
int tryckCount = 0; // global – syns i både setup och loop

void setup() {
  tryckCount = 5; // OK
}

void loop() {
  tryckCount++; // OK
}
```

**Regeln för kursen:** deklarera `const int` för pin-nummer och andra konstanter globalt, överst i filen. Deklarera tillståndsvariabler (`alarmPaslaget`, `lastState`) globalt så att de överlever mellan `loop`-varv. Tillfälliga hjälpvärden (räknare, nuvarande mätvärde) deklareras lokalt där de behövs.

### VARFÖR 'LOKAL'?

Lokala variabler försvinner när funktionen återvänder — minnet frigörs för nya variabler. Globala lever hela programmets livstid. Det är därför

```
int state = digitalRead(knappPin);
```

 kan stå inne i loop — den skapas på nytt varje varv, används, och kastas bort. `lastState` däremot måste vara global så att den överlever mellan varven.

## Kommentarer

```
// En en-rads-kommentar. Allt efter // på raden ignoreras.  
  
/*  
  En flera-rads-kommentar.  
  Allt mellan tecknen ignoreras.  
*/
```

Använd dem för att förklara **varför** något görs, inte **vad** koden gör — det ser man ändå. Bra kommentar: `// edge-detection – annars flippas larmet 10000 ggr/sek`. Dålig: `// sätt pin 13 till HIGH`.

## Arduino-specifika konstanter

Några värden Arduino-biblioteket definierar åt dig och som ni redan sett:

<code>HIGH</code> , <code>LOW</code>	De två logiska nivåerna. Används med <code>digitalWrite</code> och <code>digitalRead</code> .
<code>INPUT</code> , <code>OUTPUT</code> , <code>INPUT_PULLUP</code>	Används med <code>pinMode</code> .
<code>LED_BUILTIN</code>	Vanligtvis pin 13 — där den inbyggda LED:en sitter.
<code>A0</code> – <code>A5</code>	Analoga pinnar, används med <code>analogRead</code> .
<code>true</code> , <code>false</code>	Booleska värden.

Dessa är `#define`-s i `Arduino.h` — samma historiska arv som diskuterades tidigare. Du kan inte ändra dem, bara använda dem.

## Användbara Arduino-funktioner

Utöver de ni redan använt rakt av finns några hjälpfunktioner i Arduino-biblioteket som sparar mental ansträngning:

### MAP() — SKALA OM ETT VÄRDE

```
map(värde, frånMin, frånMax, tillMin, tillMax)
```

Skalar om ett tal linjärt från ett intervall till ett annat. Exempel — gör om fotocellens 0–1023 till PWM:s 0–255:

```
int pwmVarde = map(ljus, 0, 1023, 0, 255);
```

Eller **inverterat** — mörkt ger starkt ljus:

```
int pwmVarde = map(ljus, 0, 1023, 255, 0);
```

Mycket användbar i hackathonen när sensor-värden ska styra PWM-utgångar.

### CONSTRAIN() — KLIPP TILL ETT INTERVALL

```
int sakert = constrain(värde, minsta, största);
```

Om `värde` är under `minsta` → returnera `minsta`. Om över `största` → returnera `största`. Annars oförändrat. Användbart tillsammans med `map()` för att vara säker på att resultatet aldrig går utanför ett önskat intervall.

### MILLIS() — TID SEDAN UPPSTART

```
unsigned long nu = millis();
```

Returnerar antal millisekunder sedan Arduinon fick ström. Till skillnad från `delay()` blockerar `millis()` inte loopen — den är grunden för ”non-blocking” kod som kan göra flera saker samtidigt. Används sällan i kursen men är nästa steg för den som vill djupare.

## Vanliga kompilatorfel

expected ';' before...

Glömt semikolon på föregående rad. IDE:n pekar på **nästa** rad — felet är raden innan.

'foo' was not declared in this scope

Stavfel, eller variabel används utanför sitt scope. Kontrollera att variabeln är global eller lokal där den används.

expected '\}' at end of input

Glömt stänga ett block. Räkna måsvingarna — varje `{` måste ha en `}`.

left operand of ',' has no effect

Du har skrivit `,` istället för `;` mellan två uttryck.

```
invalid operands of types 'const char*' and 'int'
```

Du har försökt göra matematik på en sträng. Kontrollera vad du skriver in i operatorn.

En uttömmande genomgång av felmeddelanden finns i Bilaga B.

## BILAGA B

# Felmeddelanden

---

*Hur du läser Arduino-IDE:ns gnäll, och vad det faktiskt betyder.*

## Varför meddelandena ser kryptiska ut

Arduino-IDE:n använder en riktig C++-kompilator under huven (`avr-g++`). Den är ett verktyg för proffs som skrivit miljoner rader kod och accepterat att felmeddelanden ibland ser akademiska ut. De är långa, tekniska, och verkar peka på fel plats — men det finns logik bakom dem.

**Grundregeln:** felmeddelanden pekar på **den rad där kompilatorn insåg att något var fel**, inte nödvändigtvis **den rad där du gjorde misstaget**. Om du glömmer ett semikolon på rad 10 kommer kompilatorn klaga på rad 11.

**Andra grundregeln:** läs alltid det **första** felmeddelandet, inte det sista. Senare fel är ofta följdfel från det första.

## Kompilatorsfel du kommer stöta på

`EXPECTED ';' BEFORE ...`

Det vanligaste felet. Du har glömt ett semikolon.

```
digitalWrite(LED_BUILTIN, HIGH) // saknar ;
delay(1000);                    // kompilatorn klagar HÄR
```

Kompilatorn läste rad 1, förväntade sig `;` men hittade `delay` istället. Felet pekas ut på rad 2, men fixa rad 1.

`'VARIABEL' WAS NOT DECLARED IN THIS SCOPE`

En variabel används utan att ha deklarerats — ofta stavfel, eller variabeln är lokal och du försöker nå den någon annanstans.

```
void setup() {
  int temp = 5;
}

void loop() {
  Serial.println(temp); // FEL: temp finns bara i setup()
}
```

Fixa genom att flytta `int temp = 5;` till global nivå (överst i filen), eller deklarerar en ny `temp` i `loop`.

Eller — oftast — stavfelet. Två varianter av samma fel:

```
digitalWrit(13, HIGH); // FEL: saknar 'e' på slutet
digitalwrite(13, HIGH); // FEL: fel skiftläge på 'w'
```

Rätt är `digitalWrite` — med **stort W och e på slutet**. Arduino är **skiftlägeskänsligt**: `digitalwrite` och `DigitalWrite` är olika ord i kompilatorns ögon. Det vanligaste felet är att tappa `e` ;: det näst vanligaste är att glömma skiftläget.

#### EXPECTED '}' AT END OF INPUT

Du har glömt att stänga ett block någonstans i filen. Räkna alla `\{` och se till att det finns lika många `\}`.

#### HITTA OBALANSERADE KLAMMER

Arduino-IDE:n markerar matchande klammer när du klickar på en. Klicka på ett `\{` — den motsvarande `\}` ska markeras automatiskt. Om det inte finns en, har du hittat problemet.

#### 'LED\_BUILTIN' WAS NOT DECLARED IN THIS SCOPE (OVANLIGT)

Brukar bero på att `#include <Arduino.h>` saknas — men det inkluderas automatiskt av Arduino IDE. Ser du det här felet har du sannolikt skapat en fil manuellt utanför en vanlig `.ino`-sketch. Starta en ny sketch från IDE:n istället.

#### VOID VALUE NOT IGNORED AS IT OUGHT TO BE

En funktion som inte returnerar något (`void`) har använts som om den gav tillbaka ett värde:

```
int x = delay(100); // FEL: delay returnerar void
```

Rätt är att bara anropa den:

```
delay(100);
```

**ASSIGNMENT OF READ-ONLY VARIABLE**

Du har försökt ändra värdet på en `const` variabel:

```
const int ledPin = 13;
ledPin = 12;           // FEL: const går inte att ändra
```

Antingen: ta bort `const` om värdet **ska** kunna ändras. Eller: hitta den riktiga variabeln du menade att ändra.

**INVALID OPERANDS OF TYPES ... TO BINARY 'OPERATOR'**

Du har blandat inkompatibla typer i ett uttryck:

```
int x = "hej" + 5;     // FEL: sträng + int
```

Arduino är strikt med typer. Du kan inte addera ett tal till en textsträng rakt av. Fixa genom att tänka igenom vad du faktiskt vill göra — oftast menade du någonting helt annat.

**TOO MANY ARGUMENTS TO FUNCTION**

Du har skickat in fler argument än funktionen förväntar:

```
digitalWrite(LED_BUILTIN, HIGH, 100); // FEL: digitalWrite tar bara 2
argument
```

Kolla funktionens signatur. Oftast är det `delay` du egentligen ville ha på slutet:

```
digitalWrite(LED_BUILTIN, HIGH);
delay(100);
```

## Upload-fel (inte kompileringsfel)

Här kör kompilatorn klart, men programmet kan inte överföras till Arduinon.

**AVRDUDE: STK500\_GETSYNC(): NOT IN SYNC**

IDE:n kan inte prata med Arduinon. Fyra vanliga orsaker:

1. **Fel port i Tools** → **Port**. Välj den port som heter något med "Arduino" eller "usbserial".
2. **Fel board i Tools** → **Board**. Välj "Arduino Uno".
3. **USB-kabeln är en laddkabel, inte en datakabel**. Arduino Uno R3 använder en USB Type B-kontakt (den kvadratiska, "printer-style"). Kontrollera att kabeln faktiskt överför data — många billiga kablar är bara ström.
4. **Arduinon är inte ansluten till datorn**. Låter självklart, men händer.

**SERIAL PORT 'COMX' NOT FOUND**

Porten försvann. Koppla ur och i Arduinon, vänta några sekunder, välj porten igen i Tools → Port.

### PORT BUSY ELLER PORT IN USE

Serial Monitor är öppet och blockerar porten. Stäng Serial Monitor innan du uppladdar. IDE:n öppnar den automatiskt igen efter uppladdningen.

## Runtime-beteenden (inte fel — men mystiska)

Ibland kompilerar programmet, laddas upp utan problem, men beter sig inte som du väntar. Här är några klassiker:

### “PROGRAMMET GÖR INGENTING”

Vanligaste orsaker:

1. Du använder fel pinne. Kontrollera att kabeln faktiskt sitter i pin 13 och inte pin 12.
2. Komponenten är inte ansluten. Lös kabel? Har LED:en fallit av breadboarden?
3. Polaritet fel. LED:en vänd åt fel håll? Resistorn saknas?
4. `pinMode` saknas. Alla utgångar måste sättas som `OUTPUT` i `setup`, alla ingångar som `INPUT` eller `INPUT_PULLUP`.

### “PROGRAMMET GÖR NÅGOT MEN INTE DET JAG VILL”

Lägg in `Serial.println()` överallt där du misstänker att det går fel. Printa villkorsvariabler, mätvärden, vilken gren av if/else som körs. Arduinon är inte mystisk — den är bara tyst. Gör den pratig.

```
Serial.print("larmPaslaget=");  
Serial.print(larmPaslaget);  
Serial.print(" ljus=");  
Serial.println(ljus);
```

### “BUZZERN TJUTER KONSTANT”

En av grenarna i din if/else skriver `HIGH` men ingen återställer till `LOW`. Se till att **varje** gren skriver både `HIGH` och `LOW` på buzzern.

### “KNAPPEN GER FLERA TRYCK ÅT GÅNGEN”

Knappens metallblad studsar fysiskt några millisekunder. `delay(10)` eller `delay(50)` i slutet av loopen är enklaste formen av debounce och **räcker** för att släta över studsens — men bara om du också har edge-detection på knappen. Utan flank-mönstret (`state == LOW && lastState == HIGH`) toggas `larmPaslaget` hundratals gånger per tryck, vilket inget delay i världen räddar. Modul 3-sektionen ”Reagera på flanken” är icke förhandlingsbar för larmet.

### “SERIAL MONITOR ÄR TOM”

Tre saker att kontrollera:

1. `Serial.begin(9600)`; finns i `setup`?
2. Baud rate i Serial Monitor står på 9600 (matchar siffran i `Serial.begin`)?
3. Skriver du faktiskt till Serial någonstans (`Serial.print` eller `Serial.println`)?

### "SERIAL MONITOR VISAR KONSTIG TEXT"

Baud rate matchar inte. Kontrollera att både `Serial.begin()` och Serial Monitor står på samma siffra. 9600 är kursens standard.

### "LED-STYRKAN ÄNDRAS INTE MED ANALOGWRITE"

Du använder en pinne som inte stödjer PWM. Kolla att det finns `~` framför pin-numret på Arduino-kortet. Giltiga PWM-pinnar på Uno: **3, 5, 6, 9, 10, 11**.

## Felsöknings-checklista

När något inte fungerar, gå igenom i ordning:

1. **Läs felmeddelandet högt.** Ofta räcker det.
2. **Kontrollera första felmeddelandet**, inte det sista.
3. **Kolla pin-numren.** Stämmer de med vad du kopplat?
4. **Kolla polariteten.** LED, RGB-LED, buzzer — alla har riktning.
5. **Kolla att kabeln sitter i rätt hål.** "Samma rad på breadboarden" är en favoritbugg.
6. **Printa allt.** Lägg `Serial.println` i varje gren av varje if/else och läs vad som händer.
7. **Ladda om Arduino IDE.** I sällsynta fall får IDE:n hicka och måste startas om.
8. **Fråga någon.** Inte kursledaren — grannen. Två par ögon hittar fel snabbare än ett.

### FELSÖKNINGSFILOSOFI

När något inte fungerar är det frestande att **gissa**. Det är näst intill alltid fel strategi. **Ta reda på** — printa värden, kolla polariteten, läs felmeddelandet. Varje gång du fixar en bugg genom gissning istället för att verifiera, har du bara haft tur. Verifiera, vinn.

## BILAGA C

# Komponentreferens

*Snabbreferens för varenda bit ni kopplat i kursen.*

Denna bilaga är en pek-och-slå-upp-referens. Varje komponent får en kort beskrivning, pinout/polaritet om det spelar roll, vanliga fel, och hur den ska kopplas in mot Arduinon.

## LED (standard, genomskinlig)



*Lysdiod med markerad anod (+, långt ben) och katod (-, kort ben med platt kant).*

En lysdiod. Skickar man ström den rätta vägen lyser den. Fel väg: ingenting. För mycket ström: den brinner upp permanent.

<b>Polaritet</b>	Långt ben = anod (+). Kort ben med platt kant = katod (-).
<b>Max ström</b>	Ca 20 mA. Över det → bränns upp.
<b>Framspänningsfall</b>	Röd ≈ 1,8–2 V. Grön/gul ≈ 2–2,2 V. Blå/vit ≈ 3–3,4 V. Se Bilaga F.
<b>Seriemotstånd</b>	Alltid. 220 Ω är ett säkert allround-val för 5 V.
<b>Arduino-koppling</b>	Pin 13 → anod. Katod → resistor → GND.

**Vanliga fel:** LED kopplad baklänges (lyser inte, men går inte sönder), resistor saknas (lyser en kort stund, sedan bränns upp), benet ligger i fel hål på breadboard.

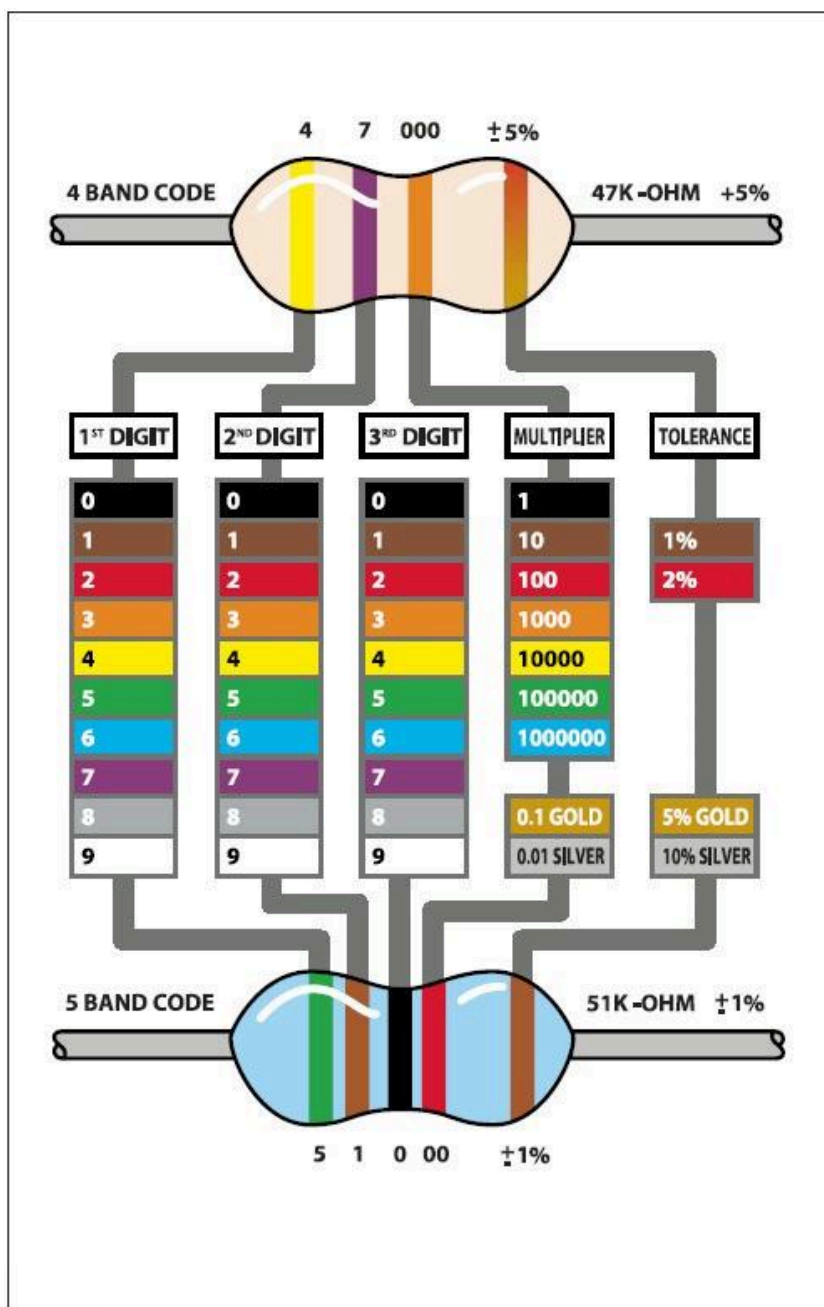
## Resistor



Resistor med färgband. Läs från den ände där banden ligger tätast.

Passiv komponent som ”bromsar” ström. Inget att lysa, inget att blinka — men den är ofta det som skiljer en fungerande krets från en brunnen komponent.

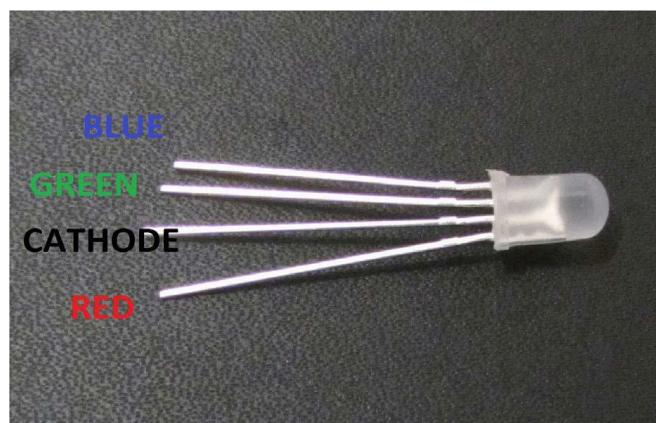
<b>Polaritet</b>	Ingen. Kopplas hur som helst.
<b>Värden i kittet</b>	220 $\Omega$ , 1 k $\Omega$ , 10 k $\Omega$ (och några varianter däremellan)
<b>Avläsning</b>	Färgband. Räkna först <b>hur många band</b> din resistor har — kittet innehåller både 4-bands (normal precision) och 5-bands (1 % precision).
<b>4-band</b>	första två siffror · multiplikator · tolerans. 220 $\Omega$ = röd-röd- <b>brun</b> -guld.
<b>5-band</b>	tre siffror · multiplikator · tolerans. 220 $\Omega$ = röd-röd-svart- <b>svart</b> -brun.
<b>1 k<math>\Omega</math> (4-band)</b>	brun · svart · röd · guld
<b>1 k<math>\Omega</math> (5-band)</b>	brun · svart · svart · brun · brun
<b>10 k<math>\Omega</math> (4-band)</b>	brun · svart · orange · guld
<b>10 k<math>\Omega</math> (5-band)</b>	brun · svart · svart · röd · brun
<b>Osäker?</b>	Mät med multimeter i kontinuitet/ $\Omega$ -läge. 220 $\Omega$ ger exakt "220" på skärmen. Alltid säkrast när du tvivlar.



Färgkodstabell. För 4-band: digit-digit-multiplikator-tolerans. För 5-band (1 %-precision): digit-digit-digit-multiplikator-tolerans. Räkna banden innan du börjar dekodra.

**Tabellen för färgkoder** sitter också inuti locket på kittet. Du är välkommen att använda den eller mäta med multimeter. Med tiden lär man sig de vanligaste värdena utan att tänka.

## RGB-LED (common cathode)



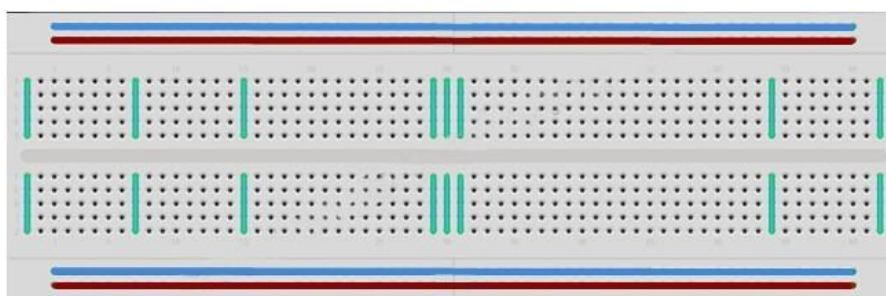
RGB-LED med de fyra benen markerade. Från platta sidan: röd, katod (längst), grön, blå.

Fyra ben, tre kanaler (röd, grön, blå) som delar en gemensam katod. Kittets variant är **common cathode** — det längsta benet går till GND.

<b>Pin-ordning från platta sidan</b>	röd · katod · grön · blå
<b>Katoden</b>	Andra benet från platta sidan. Längst av de fyra. Till GND.
<b>De tre andra</b>	Via varsin 220 $\Omega$ -resistor till PWM-pinne.
<b>Kursens pinnar</b>	R $\rightarrow$ D6, G $\rightarrow$ D5, B $\rightarrow$ D3.
<b>Varför PWM?</b>	För att kunna variera ljusstyrkan per kanal (0–255).

**Vanligt fel:** kopplad som common anode (med gemensam + istället för GND). Då fungerar inget. Om du bara får en färg lysa: ett av benen är i fel hål eller saknar resistor.

## Breadboard



Breadboard ovanifrån med interna förbindelser markerade: röda/blåa power rails längs sidorna, gröna kolumn-noder i mitten, gapet som bryter förbindelsen.

Plastbit med dolda metallklämmor som låter dig koppla ihop komponenter utan att löda.

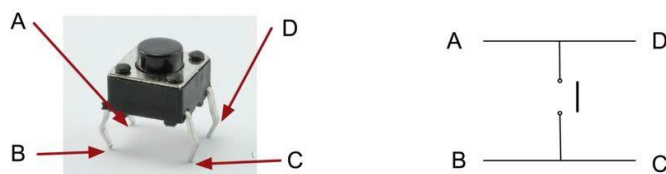
<b>Horisontella rader i mitten</b>	Fem hål i rad är en nod. Raden bredvid är en egen nod.
<b>Gapet i mitten</b>	Bryter förbindelsen mellan övre och nedre halvan.

**Power rails (+/-) längs sidorna** Hela raden är en enda lång nod. Används för att dela ut 5 V och GND.

**Hål diameter** Passar genomgångshåls-komponenter och DuPont-kablar (0,6 mm).

**Vanligaste nybörjarfelet:** ett komponentben och en kabel ligger i olika rader. Kolla först att de **faktiskt sitter på samma rad**.

## Knapp (tactile switch)



*Tactile switch — fyra ben, men internt bara två noder (A–D hopkopplade, B–C hopkopplade). Tryckning sluter de två noderna.*

En liten mekanisk brytare med fyra ben. När du trycker ner den kortsluter två par av benen.

**Pinnar** Fyra ben, men elektriskt bara två noder: A–D är internt ihopkopplade, B–C likaså. Tryckning kortsluter båda paren.

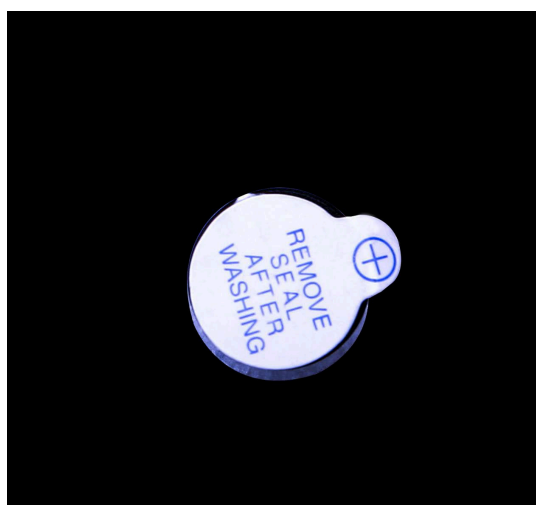
**Koppling i kurs** Ena benet → D9, andra benet → GND. Ingen extern resistor.

**Läs med** `digitalRead(pin)` efter `pinMode(pin, INPUT_PULLUP)`.

**Tryckt = LOW** På grund av inverterad logik med INPUT\_PULLUP.

**Studs** Fysisk studs några millisekunder. Lägg `delay(10)` eller `delay(50)` för enkel debounce.

## Active buzzer



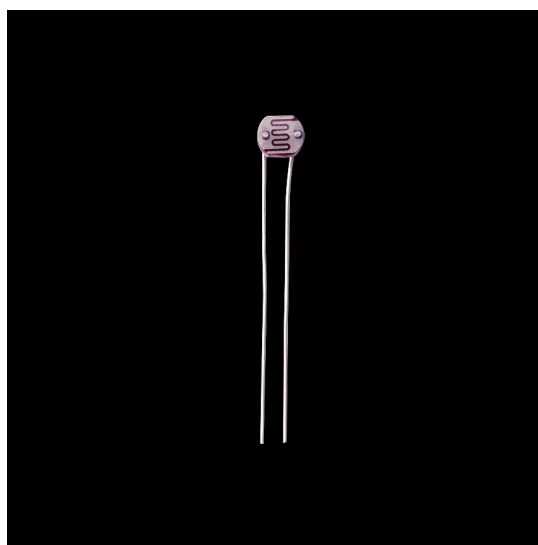
*Active buzzer — svart cylinder med inbyggd oscillator. Klisterlappen på ovsidan **stannar kvar**.*

En liten svart cylinder som ger ifrån sig ljud när du skickar ström genom den. **Active** betyder att den har en inbyggd oscillator — `digitalWrite(pin, HIGH)` ger direkt pip, ingen `tone()` behövs.

<b>Polaritet</b>	Långa benet = plus (markerat med + på ovansidan). Kort ben = minus.
<b>Koppling i kurs</b>	+ -ben (höger, markerat) → D12 , - -ben → GND -skenan på breadboarden tillsammans med knappen.
<b>Resistor</b>	Behövs inte. Buzzern är självreglerande.
<b>Klisterlappen</b>	Stannar på. "Remove after washing" är en dämpare från tillverkningsprocessen.
<b>Frekvens</b>	Fast ( 2 kHz). Inte ställbar.

**Varning:** passive buzzer (blå, låg) finns också i kittet och ser liknande ut. Den kräver `tone(pin, frequency)` för att låta. I kursen använder vi active buzzer — svart, hög, med klisterlapp.

## Fotocell (LDR)



Fotocell (LDR) — rund skiva med karakteristiskt gult snake-eye-mönster på ovansidan.

”Light-dependent resistor”. En resistor vars värde ändras beroende på ljuset som träffar den.

<b>Polaritet</b>	Ingen. Kopplas hur som helst.
<b>Motståndsområde</b>	Ca 50 k $\Omega$ i mörker, ca 500 $\Omega$ i starkt ljus.
<b>Måste ha partner</b>	Ingår i en spänningsdelare — behöver en fast resistor (1 k $\Omega$ ) för att bilda mellanpunkten.
<b>Koppling i kurs</b>	5 V → fotocell → A0 → 1 k $\Omega$ → GND.

**Läses med** `analogRead(A0)` — ger 0–1023.

**Mätvärden (ungefärliga, med 1 k $\Omega$  mot GND):** mörker  $\approx$  20–100, rumsljus  $\approx$  150–400, lampa nära  $\approx$  500–700, ficklampa direkt  $\approx$  700–900. Kalibrera alltid själv.

## Tilt-sensor



*Tilt ball switch — cylindrisk kapsel med lös metallkula inuti.*

En liten cylinder med en lös metallkula inuti. I upprätt läge rör kulan bara ett ben — kretsen är bruten. Lutad: kulan rullar till det andra benet och kortsluter dem.

<b>Polaritet</b>	Ingen för själva funktionen. Men kopplas som en knapp: ena benet till pinne, andra till GND.
<b>Digital trots att det är en sensor</b>	Läses med <code>digitalRead</code> , inte <code>analogRead</code> . Bara två lägen.
<b>Koppling i kurs</b>	Ena benet $\rightarrow$ D2, andra $\rightarrow$ GND. Rakt i Arduino-headers.
<b>Pinmode</b>	<code>pinMode(tiltPin, INPUT_PULLUP)</code> .
<b>Lutad = LOW</b>	Inverterad logik från INPUT_PULLUP.
<b>Studs</b>	Kulan bouncar. <code>delay(50)</code> efter läsningen räcker som debounce.

## Vridpotentiometer (10 kΩ) — demonstrationskomponent

Kittet innehåller en **blå vridpotentiometer** på 10 kΩ. Ni kopplar den inte själva i kursen — men i Modul 4 visar läraren den som **fysisk modell av en spänningsdelare**, för att göra det konkret vad fotocellen sedan gör automatiskt.

<b>Vad det är</b>	Ett "långt motstånd" där en glidande arm (wiper) kan ställas var som helst längs det inre motståndsmaterialet.
<b>Pinnar</b>	Tre ben. Två yttre = ändarna av motståndsspåret. Mittben = wipern.
<b>Demo-koppling</b>	Yttre ben → +5 V och GND. Mittben → A0. Vrid → A0 sveper 0–1023 jämnt.
<b>Maxvärde</b>	10 kΩ totalt mellan de två yttre benen. Wipern delar upp denna resistans i två andelar.
<b>Läses med</b>	<code>analogRead(A0)</code> . Ger samma 0–1023 som fotocell-kopplingen.

Varför den ligger utanför hands-on-laben: en pot löser samma uppgift som en knapp i många nybörjarprojekt (välja ett värde) men introducerar mekanik som tar fokus från koden. Den är dock perfekt som **brygga** till spänningsdelar-konceptet — wipern är mätpunkten, ni kan se den flyttas med handen.

**Vill ni leka mer:** prova att ersätta fotocellen i Modul 4 med pot:en och styra LED-ljusstyrkan med vridning istället för ljus. Allt i Modul 4-koden fungerar oförändrat.

## Arduino Uno R3

Själva styrkretsen. En mikrocontroller i ett "development board"-format med inbyggd USB-kontroller, spänningsreglering och stift-kontakter.

<b>Processor</b>	ATmega328P (16 MHz, 8-bit)
<b>Minne</b>	32 KB flash (din kod), 2 KB SRAM (variabler), 1 KB EEPROM (persistent)
<b>Digital-pinnar</b>	D0–D13, varav <b>D3, D5, D6, D9, D10, D11</b> stödjer PWM (markerade med ~)
<b>Analoga pinnar</b>	A0–A5, kan läsas med <code>analogRead</code> (0–1023).
<b>Strömförsörjning</b>	USB (5 V) eller externt via DC-jacket (7–12 V → regleras till 5 V internt).
<b>5 V-pin</b>	Ger ut 5 V vid USB-strömförsörjning. Totala budgeten är ca 450 mA efter att kortets egna komponenter tagit sitt. Mer än så och USB-portens polyfuse (500 mA) bryter.
<b>3,3 V-pin</b>	Ger ut 3,3 V. Användbar för vissa sensorer. Max 50 mA.
<b>GND</b>	Tre stift, alla elektriskt samma punkt. Vilken som helst fungerar.

<b>Reset-knappen</b>	Startar om din sketch från början. Användbar om den krashar.
<b>Inbyggd LED</b>	Pin 13, alltid kopplad till en intern LED. Använd <code>LED_BUILTIN</code> i koden.

## Kabelbröderna — DuPont-kablar

De färgade trådarna du använder för att koppla mellan breadboard och Arduino.

<b>M-M (hane-hane)</b>	Stift i båda ändar. För breadboard-till-breadboard eller breadboard-till-header.
<b>M-F (hane-hona)</b>	Ena änden stift, andra änden hylsa. För Arduino-header till breadboard, eller till direkta komponenter som tilt-sensor.
<b>F-F (hona-hona)</b>	Hylsa i båda ändar. Används sällan i kursen.
<b>Typisk längd</b>	10–20 cm. Inte kritiskt men kortare är snyggare.

## 74HC595 skiftregister — den komponenten vi inte använder

Kittet innehåller **en** komponent som kursen inte rör: ett 74HC595-skiftregister. Det är en 16-bens IC (integrerad krets) som låter Arduinon styra 8 LED-utgångar med bara 3 pinnar.

<b>74HC595 skiftregister</b>	16-bens DIP-IC. Tre styripinnar ( <code>data</code> , <code>clock</code> , <code>latch</code> ) kan styra 8 utgångar. Principen: skiftregistret "kommer ihåg" en 8-bits sekvens du matar in en bit i taget, och håller dem alla samtidigt på sina utgångar.
------------------------------	---

Det är en utmärkt första fördjupning efter kursen — prova att blinka åtta LED:ar i ett löpande mönster. Sök efter "74HC595 shift register Arduino" för exempel.

## Vill du ha fler komponenter?

Kittet är medvetet minimalt — de komponenter ni byggt med under fem träffar är i princip precis de som ryms. Om ni vill gå vidare:

- **Utökningskit** (Super Starter Kit, Most Complete Kit) innehåller saker som servomotor, LCD-display, IR-mottagare, temperatursensor, ultraljudsavståndsmätare, m.fl.
- **Lösa komponenter** från Kjell & Company, Electrokit eller liknande när ni har ett specifikt projekt i åtanke. Ofta mer prisvärt än att köpa ett nytt kit.
- **Online-tutorials** — Hackster.io och Instructables förklarar nästan alltid vad de använder och var man får tag på det.

## BILAGA E

# Säkerhet

---

*De saker som gör att du inte bränner dig, LED:en, eller Arduinon.*

## Ofarligt för dig — farligt för utrustningen

Spänningen du jobbar med på kursen är **5 V**. Det är samma spänning som en USB-kabel. **För människor är det helt ofarligt** — du kan hålla båda ledningarna mellan tummen och pek fingret och känna ingenting alls. Du jobbar inte på vägguttags-nivå.

Men **din utrustning** — laptopen, Arduinon, komponenterna — är inte lika robust. En kortslutning mellan **+5 V** och **GND**, en LED utan resistor, en pinne som dras över sin maxgräns — alla är saker som kan stänga av USB-porten (bra) eller rosta komponenter (mindre bra). Fem små saker att veta:

### 1. LED:en är riktad — kontrollera polariteten

Skickar du ström åt fel håll genom en LED: den lyser inte. Skickar du för mycket ström åt rätt håll: den brinner upp på mindre än en sekund.

- **Långt ben = plus (anod)**. Mot signal-pinnen.
- **Kort ben + platt kant på plastkuddens rand = minus (katod)**. Mot GND.
- **Seriemotstånd är obligatoriskt**. 220  $\Omega$  duger för 5 V.

Utan resistor går mer än 20 mA genom LED:en, och den hinner lysa i högst några sekunder innan den bränns upp permanent. Den är inte farlig för dig — men du får byta. Kittet har ett par extra, men regeln är enkel: **aldrig en LED utan seriemotstånd**.

### 2. Ingen pinne tål hur mycket ström som helst

Arduinons digitala pinnar har en **rekommenderad maxström på 20 mA per pinne** (absolut maximum enligt databladet är 40 mA, men det sliter på chippet). Totalt genom chippet

(alla pinnar tillsammans): absolut max är **200 mA** — håll dig klart under det. Drar du mer bränner du en liten effekttransistor inne i chippet. Det märks inte direkt — men efter ett tag slutar pinnen svara på `digitalWrite`.

**För kursen är det inget problem.** En LED via  $220\ \Omega$  drar ca 13 mA. En RGB-LED med alla tre kanaler full = ca 40 mA. En buzzer = ca 30 mA. Allt med god marginal.

**Fel att undvika:**

- En LED direkt till 5 V utan resistor → drar 25+ mA direkt, bränner pinnen eller LED:en eller båda.
- Två LED:ar parallellt på samma resistor → delar strömmen olika, en tar skada.
- Försök driva en motor direkt från pinnen → motorer drar hundratals mA, Arduinon överlever inte.

**Regel:** om du kopplar något som inte är uttryckligen beskrivet i kursen, räkna om strömmen med Ohms lag först.

### 3. Kortslut inte 5 V och GND

Om du direkt kopplar `+5 V` till `GND` med en ren kabel, börjar Arduinon dra hela strömmen den kan leverera. Kabeln blir varm; i värsta fall smälter dess isolering eller en spårledning på kortet bränns.

**Det låter allvarligare än det är.** USB-porten har en skyddskrets som bryter vid ca 500 mA. Men: kabeln blir VÄLDIGT varm, laptop-varningen poppar upp, och du får koppla ur i rätt tid.

**Vanligaste sättet att göra det av misstag:**

- En lös kabel som hamnar i GND-raden och rör plus-raden på breadboardens power rails.
- En resistor med brutna ben där ena änden rör plus och andra änden rör minus utanför kretsen.
- Förvirrande fingerplats när du sticker in en kabel.

Om du märker att Arduinon blir varm — **koppla ur USB:n direkt**. Kolla vart du senast drog en kabel.

### 4. Statisk elektricitet är en verklig sak

Torr vinterluft + nyfallen snö + syntetiska tröjor → du bygger upp några tusen volt statisk elektricitet på kroppen. När du rör Arduino-chippets pinnar med ett finger kan det ladda ur sig genom chippet och permanent skada det.

**Det är en liten risk på Arduino Uno** (den är byggd för att vara robust), men risken finns för andra IC-komponenter. I kursen är det sällsynt att detta händer, men värt att veta.

**Förebygg:** ta tag i något jordat (en metallkran, värmeelementet, laptopens USB-kontakt) innan du börjar koppla om du misstänker att du är uppladdad. Eller: **rör aldrig bara själva metall-pinnarna på chipen** — håll i plastkanterna eller i själva kretskortet.

**För kursens komponenter** spelar det mindre roll. LED, resistor, knapp, buzzer, fotocell — ingen av dem är känslig för statisk. Men IC-kretsar som 74HC595 **är**.

## Säker felsökning

När något inte fungerar är det ibland frestande att ”testa med att snabbt koppla fram och tillbaka”. **Koppla inte om medan Arduinon är strömförsörjd.** Dra ur USB:n, koppla om, koppla i igen. Det skyddar mot oväntade kortslutningar och gnistor.

### Ordning:

1. Koppla ur USB.
2. Ändra kopplingen.
3. Kontrollera att inget oväntat är kortslutet (titta, tänk efter).
4. Koppla i USB.
5. Ladda upp ny kod om det behövs.

Det tar 5 extra sekunder per iteration och sparar miljoner sekunder av felsökning.

## Och till slut: ofarligt för människan

**Säg det högt en gång till:** 5 V via USB är inte farligt för **dig**. Du kan hålla båda ledningarna, röra vid chipet med handen, droppa vatten på breadboardet (nej, bara skämt — torka upp det), och inget händer med dig. Det är komponenterna och laptopen man skyddar — inte fingrarna.

Det värsta som kan hända på kursens kompetensnivå är:

1. En LED brinner upp → du får byta till en av de andra i kittet.
2. En USB-port i laptopen stänger av p.g.a. överskydd → du kopplar i igen.
3. En Arduino slutar svara på en pinne → byt till en annan pinne och anteckna det.

Ingenting brinner. Ingen skadas. Värsta utfallet är att du får köpa en ny 29-kronors LED från Kjell & Company. **Slappna av. Bygg.**

## BILAGA F

# Ohms lag & framspänningsfall

---

*Den tyngsta bilagan. Här utreder vi varför  $220\ \Omega$  är  $220\ \Omega$  och ingenting annat.*

## Vad kapitlet handlar om

Det här är inte en rundtitt. Det är ett djupdyk. Om ni satt på träff 1 och undrade ”men **varför**  $220\ \Omega$ ?” utan att hinna fråga, är det här svaret. Om ni redan kunnat svaret i detalj, är det en repetition på ert modersmål.

Målet är att efter läsningen ska ni kunna:

1. **Räkna ut** vilken resistor en given LED behöver utan att googla.
2. **Förklara** varför en LED inte följer Ohms lag direkt.
3. **Förstå** vad som händer om resistorn är för liten eller för stor.
4. **Läsa** en spänningsdelar-koppling som fotocellens och veta varför mellanpunkten rör sig när ljuset ändras.

Allt med svenska ord, inga latinska uttryck som jag inte vill förklara.

## Ohms lag — den ultimata tre-storheten

Tre storheter som alltid hör ihop i en elektrisk krets:

**Spänning (U)** Tryckskillnaden mellan två punkter. Enhet: volt (V). Exempel: 5 V.

**Ström (I)** Hur mycket laddning som rör sig per sekund. Enhet: ampere (A). I småsensorik räknas det ofta i milliampere (mA).

**Motstånd (R)** Hur mycket ett material bromsar strömmen. Enhet: ohm ( $\Omega$ ). Exempel: 220  $\Omega$ .

## FORMELN

$$U = I \cdot R$$

I ord: **spänningen över en resistor är lika med strömmen genom den, gånger resistorns värde.**

Omformulerat för de gånger du söker annan storhet:

$$I = \frac{U}{R} \quad R = \frac{U}{I}$$

Eller med den klassiska **triangeln**. Rita en pyramid med  $U$  på toppen och  $I$  och  $R$  i botten. Håll över den storhet du vill räkna ut, så ser formeln sig själv.

$$\begin{array}{c} U \\ \text{-----} \\ I \cdot R \end{array}$$

Håll för  $U \rightarrow$  ser du  $I \cdot R$ . Håll för  $I \rightarrow$  ser du  $\frac{U}{R}$ . Håll för  $R \rightarrow$  ser du  $\frac{U}{I}$ . Fungerar **varje gång**.

## VARFÖR BOKSTÄVERNA U OCH I?

**U** kommer från tyskans ”Spannung” (spänning). **I** från franskans ”intensité du courant” (strömintensitet, Ampères terminologi).

Anglosaxiska böcker använder ibland  $V$  för spänning istället för  $U$  — samma sak, olika lokala traditioner. I Sverige och Tyskland:  $U$ . I Storbritannien och USA:  $V$ .

## FÖRSTA RÄKNEEXEMPLET

Om du har en resistor på 220  $\Omega$  och strömmen genom den är 15 mA, hur stor är spänningen över den?

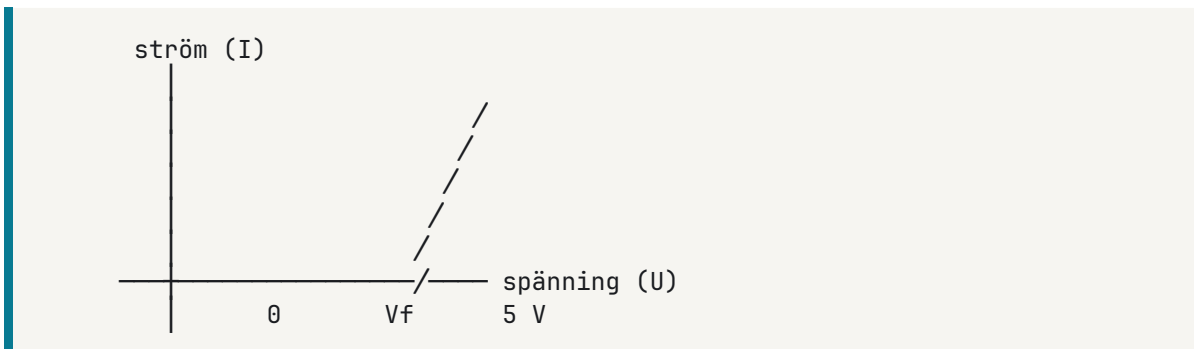
$$U = I \cdot R = 0,015 \text{ A} \cdot 220\Omega = 3,3 \text{ V}$$

Bra att komma ihåg: **ström ska omvandlas till ampere (inte milliampere) innan du räknar**. 15 mA = 0,015 A.

## Varför en LED inte följer Ohms lag

Här kommer det som knyter ihop allt. En resistor är en snäll komponent — den följer Ohms lag slaviskt. Dubbla spänningen, dubbla strömmen. Halvera spänningen, halvera strömmen. Linjärt.

En **LED är inte en resistor**. Den är en halvledar-diod med en trösklad I-V-karakteristik. På en graf av ström mot spänning ser den ut så här (ungefär):



**Upp till  $V_f$**  (framspänningen): nästan ingen ström alls. LED:en lyser inte.

**Vid  $V_f$** : LED:en ”vaknar” och börjar leda ström. Den lyser.

**Strax över  $V_f$** : ström stiger **dramatiskt** med minimal ökning i spänning. Det här är farligt — en liten ökning av spänningen ger stor ökning av strömmen.

Så vad är  $V_f$ ? Det är **framspänningsfallet** — den spänning LED:en ”kräver” för att leda ström. För en röd LED är  $V_f$  ungefär 2 V.

### KONSEKVENSEN

Om du bara kopplar en röd LED till +5 V utan resistor, försöker LED:en dra så mycket ström den kan. Den lägger 2 V över sig själv (sin  $V_f$ ) och de återstående 3 V har ingenstans att ta vägen — förutom att driva STRÖM genom LED:en. Massiv ström. Dioden bränns upp på under en sekund.

**Lösningen:** en resistor i serie som ”äter upp” de återstående 3 V:en och håller strömmen på en säker nivå.

## Räkneexemplet — hur man kommer fram till 220 $\Omega$

Krav:

- Matningsspänning: **5 V** från Arduinons pin 13.
- LED: **röd**, framspänningsfall  $\approx$  **2 V**.
- Önskad ström: **15 mA** = 0,015 A. (Säker nivå långt under LED:ens 20 mA-max.)

Tanken är: resistorn och LED:en ligger i **serie**. Strömmen genom dem är samma. Spänningen fördelas så att LED:en tar sina 2 V och resistorn tar resten. Det ska bli 5 V totalt.

$$U_{\text{resistor}} = 5 \text{ V} - 2 \text{ V} = 3 \text{ V}$$

Nu vet vi spänningen över resistorn och strömmen genom den. Ohms lag löser resten:

$$R = \frac{U}{I} = 3 \frac{\text{V}}{0,015 \text{ A}} = 200 \Omega$$

**Exakt svar: 200  $\Omega$ .** Men resistor-värden säljs bara i fasta standardstegstorlekar — E12-serien, som innehåller 100, 120, 150, 180, **220**, 270, 330, 390, 470, osv. Det närmsta tillgängliga värdet över 200 är **220  $\Omega$** .

Kolla vad strömmen faktiskt blir med 220  $\Omega$ :

$$I = \frac{5 \text{ V} - 2 \text{ V}}{220 \Omega} = 13,6 \text{ mA}$$

Lite lägre än vårt ideal på 15 mA. **Det är helt OK.** LED:en lyser fortfarande starkt och håller sig inom säkra gränser. Skillnaden är ungefär som om en lampa kördes på 85 % av sin fulla styrka — synligt, men marginellt.

Det är **därför** 220  $\Omega$  valdes. Inget gissningsval — ren matematik.

## Vad händer om resistorn är för liten?

Säg att du byter till **120  $\Omega$**  istället. Strömmen blir:

$$I = \frac{5 \text{ V} - 2 \text{ V}}{120 \Omega} = 25 \text{ mA}$$

25 mA > 20 mA (LED:ens max). Den börjar värmas upp, lyser starkare initialt, och tappar ljusstyrka efter någon minut som kiselet inuti degraderar. Efter några timmar eller dagar är den obrukbar — ljuset blir märkbart svagare, slut.

Säg att du använder **ingen resistor alls** (0  $\Omega$ ). Då begränsar bara LED:ens egen fysik strömmen. Den lägger 2 V över sig själv, men det finns inget som tar upp de andra 3 V — strömmen skenar iväg och LED:en bränns upp på **mindre än en sekund**.

### EN LED UTAN SERIEMOTSTÅND ÄR EN DÖD LED INOM FEM SEKUNDER

Låt det här vara en av de första reglerna ni lär er utantill. **Varje LED behöver en resistor i serie.** Utan undantag på kursens nivå. När ni sitter hemma och experimenterar och det finns en dag då ni är trötta och stressade och hoppar över resistorn ”bara den här gången” — det är dagen ni får köpa nya LED:ar.

## Vad händer om resistorn är för stor?

Säg att du använder **2,2 k $\Omega$**  (2200  $\Omega$ ) istället för 220. Strömmen blir:

$$I = \frac{5\text{ V} - 2\text{ V}}{2200\ \Omega} = 1,4\text{ mA}$$

Det är tio gånger **mindre** än 15 mA. LED:en lyser — men **svagt**. Ungefär som en stearinlåga i dagsljus. Du kan knappt se att den är tänd. Ingen skada, bara svag.

Om resistorn är ännu större — säg **10 k $\Omega$**  — blir strömmen 0,3 mA. LED:en lyser kanske, kanske inte, beroende på LED:ens tröskelström. Osynlig för praktiska ändamål.

Så: **för liten resistor = LED dör. För stor resistor = LED lyser svagt eller inte alls**. Mellan dem finns ett lagom intervall på säg 180–330  $\Omega$  för röda LED:er på 5 V. 220  $\Omega$  ligger mitt i mjuka delen.

## Framspänningsfall för andra färger

En viktig detalj: **olika LED-färger har olika Vf**. Det betyder att räkningen för en grön eller blå LED ger ett annat resistor-värde.

<b>Röd</b>	Vf $\approx$ 1,8 – 2,0 V. Resistor för 15 mA på 5 V: runt 200 $\Omega$ → välj 220.
<b>Gul</b>	Vf $\approx$ 2,0 – 2,2 V. Nästan samma som röd. 220 $\Omega$ fungerar.
<b>Grön</b>	Vf $\approx$ 2,0 – 2,2 V (klassisk grön) eller 3,0 – 3,2 V (modern "ren grön"). 220 $\Omega$ för den klassiska.
<b>Blå</b>	Vf $\approx$ 3,0 – 3,4 V. Resistor: (5 – 3,2) / 0,015 $\approx$ 120 $\Omega$ . Men 220 $\Omega$ fungerar också, bara något svagare.
<b>Vit</b>	Vf $\approx$ 3,0 – 3,4 V (samma som blå, eftersom vita LED:er är blåa med fosforcoating). 220 $\Omega$ duger.

**Varför 220  $\Omega$  är ett säkert allroundval:** det är lagom för alla färger i kittet. Röda LED:ar hamnar på 13 mA (säker). Blåa LED:ar hamnar på 7 mA (något svaga men fungerar). Ingen färg får för mycket ström.

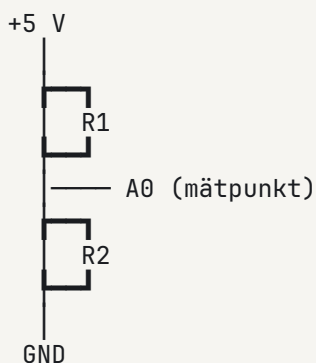
Om du vill optimera: räkna om med varje LED:s Vf och välj närmaste E12-värde. För kursen: 220  $\Omega$  överallt, klart.

## Spänningsdelaren — ohms lag för två resistorer

I Modul 4 byggde ni en **spänningsdelare** för att läsa av fotocellens motstånd. Så här fungerar den på papper.

### KOPPLINGEN

Två resistorer i serie mellan +5 V och GND. Mätpunkten är **mellan** dem.



Strömmen genom hela serien är samma (det är vad "serie" betyder):

$$I = \frac{5 \text{ V}}{R_1 + R_2}$$

Spänningen vid mätpunkten, mätt mot GND, är spänningen **över R2**:

$$V_{\text{mid}} = I \cdot R_2 = \left( \frac{R_2}{R_1 + R_2} \right) \cdot 5 \text{ V}$$

Detta är **spänningsdelar-formeln**. Den säger: andelen av inkommande spänning som hamnar på mätpunkten är samma som andelen  $R_2$  i den totala resistansen.

### EXEMPEL 1 — TVÅ LIKA STORA RESISTORER

Om  $R_1 = R_2 = 1 \text{ k}\Omega$ :

$$V_{\text{mid}} = \left( \frac{1000}{1000+1000} \right) \cdot 5 = 0,5 \cdot 5 = 2,5 \text{ V}$$

Mellanpunkten ligger exakt halva. Intuitivt vettigt: lika bromsning på båda sidorna = spänningen delas lika.

### EXEMPEL 2 — FOTOCELL I MÖRKER

Fotocell i mörker  $\approx 50 \text{ k}\Omega$ . Fast resistor =  $1 \text{ k}\Omega$ . Fotocellen är R1 (över),  $1 \text{ k}\Omega$  är R2 (under):

$$V_{\text{mid}} = \left( \frac{1000}{50000+1000} \right) \cdot 5 = 0,0196 \cdot 5 \approx 0,098 \text{ V}$$

**A0 visar ett väldigt lågt värde.** Arduinons analog-till-digital-omvandlare mäter  $0,098 \text{ V}$  → ungefär 20 på 0–1023-skalan.

### EXEMPEL 3 — FOTOCELL I STARKT LJUS

Fotocell i starkt ljus  $\approx 500 \Omega$ . Fast resistor =  $1 \text{ k}\Omega$ :

$$V_{\text{mid}} = \left( \frac{1000}{500+1000} \right) \cdot 5 = 0,667 \cdot 5 \approx 3,33 \text{ V}$$

**A0 visar cirka 682** ( $3,33 \text{ V} / 5 \text{ V} \times 1023 \approx 682$ ).

### SÅ VARFÖR PRATAR NI OM SIFFROR SOM 150 / 400 / 900?

Verkligt rumsljus ligger någonstans mellan mörk och strålande sol. Arduinos ADC är 10-bits (1024 steg) vilket innebär runt 5 mV upplösning. I praktiken:

Mörker, hand över cellen	A0 $\approx$ 20–100
Rumsljus, taklampa	A0 $\approx$ 150–400
Direkt bordslampa	A0 $\approx$ 500–700
Ficklampa direkt	A0 $\approx$ 700–900
Midsommarsol direkt	A0 $\approx$ 900 – 1023 (mättad)

Dessa är **riktvärden**. Din exakta fotocell, din breadboard, ditt rum — alla påverkar. Kalibrera alltid med Serial Monitor innan du sätter en tröskel i koden.

## Serie-resistorer, parallell-resistorer — snabbregler

För den som vill veta lite mer:

**Två resistorer i serie:** den totala resistansen är summan.

$$R_{\text{total}} = R_1 + R_2$$

**Två resistorer parallellt** (ena änden ihop, andra änden ihop): den totala resistansen är **mindre** än den minsta. Formeln:

$$R_{\text{total}} = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

Ett specialfall värt att komma ihåg: två **lika stora** resistorer parallellt = halva värdet. Två 220  $\Omega$  parallellt = 110  $\Omega$ .

I kursen använder vi bara resistorer i serie. Men när ni googlar Arduino-projekt online kommer ni stöta på parallell-kopplingar också — nu vet ni vad de betyder.

## Sammanfattning — det mest centrala

1. **Ohms lag:**  $U = I \cdot R$ . Tre variabler, en formel, tre omformuleringar.
2. **En LED kräver resistor i serie.** Alltid. Varje gång.
3. **Räkningen för 5 V + röd LED + 15 mA:**  $(5 - 2) / 0,015 = 200 \Omega \rightarrow$  välj 220  $\Omega$ .
4. **Olika färger = olika Vf**, men 220  $\Omega$  funkar som allroundval på 5 V.

5. **Spänningsdelaren** delar upp en spänning mellan två seriekopplade resistorer. Mellanpunktens spänning bestäms av förhållandet mellan resistorerna.

Och det viktigaste: **när något inte fungerar i en krets är oftast svaret ”räkna om”**. Inte ”gissa igen”.