MODUL 1

LED & krets

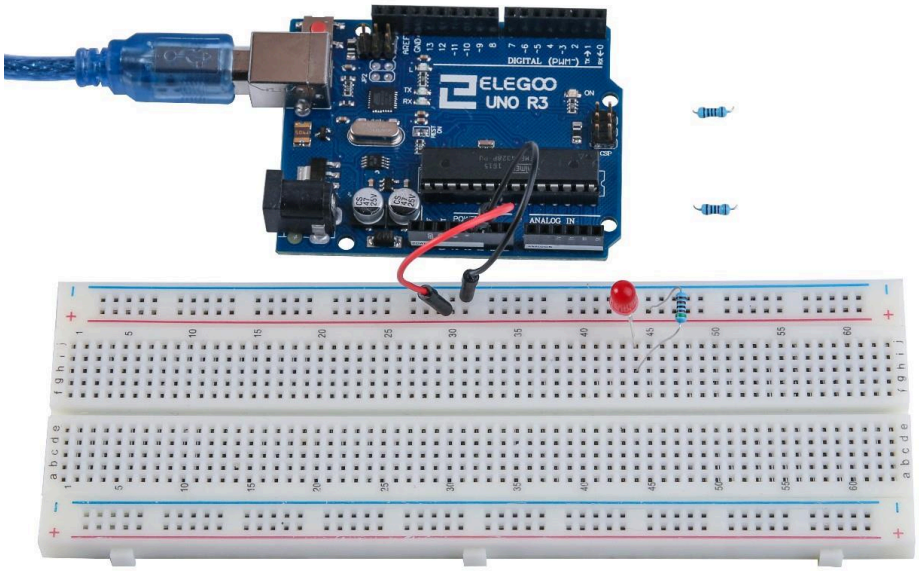
Digital output, Ohms lag, och en lampa som faktiskt blinkar.

Vad du lärde dig idag

Den första träffen handlade om att få något — vad som helst — att hända. En lampa som tänds, släcks, och blinkar i en rytm du själv bestämmer. Bakom det där enkla resultatet ligger fem idéer som hela resten av kursen vilar på.

- **En krets är en ring**. Strömmen måste lämna en pinne på Arduinon, passera genom dina komponenter, och komma tillbaka till GND. Bryts ringen händer ingenting.
- **Spänning och ström hör ihop men är inte samma sak**. Spänning är tryckskillnad (volt), ström är flöde (ampere). Ohms lag binder ihop dem via motstånd: $U = I \cdot R$.
- **En LED har polaritet**. Långt ben är plus (anod), kort ben med platt kant är minus (katod). Kör du den baklänges lyser den inte — men går inte sönder.
- **En LED behöver en resistor i serie**. Utan något som bromsar strömmen brinner dioden upp. En 220 Ω-resistor är ett säkert allround-val för 5 V-matning.
- **`digitalWrite` styr en pinne mellan två lägen**: `HIGH` (5 V ut) och `LOW` (0 V). Det är allt som händer i `Blink` under huven.

Ni kopplade blink-kretsen på breadboarden, laddade upp IDE:ns färdiga `Blink`-exempel, och ändrade sedan koden för att få er egen rytm — många valde SOS i morse.

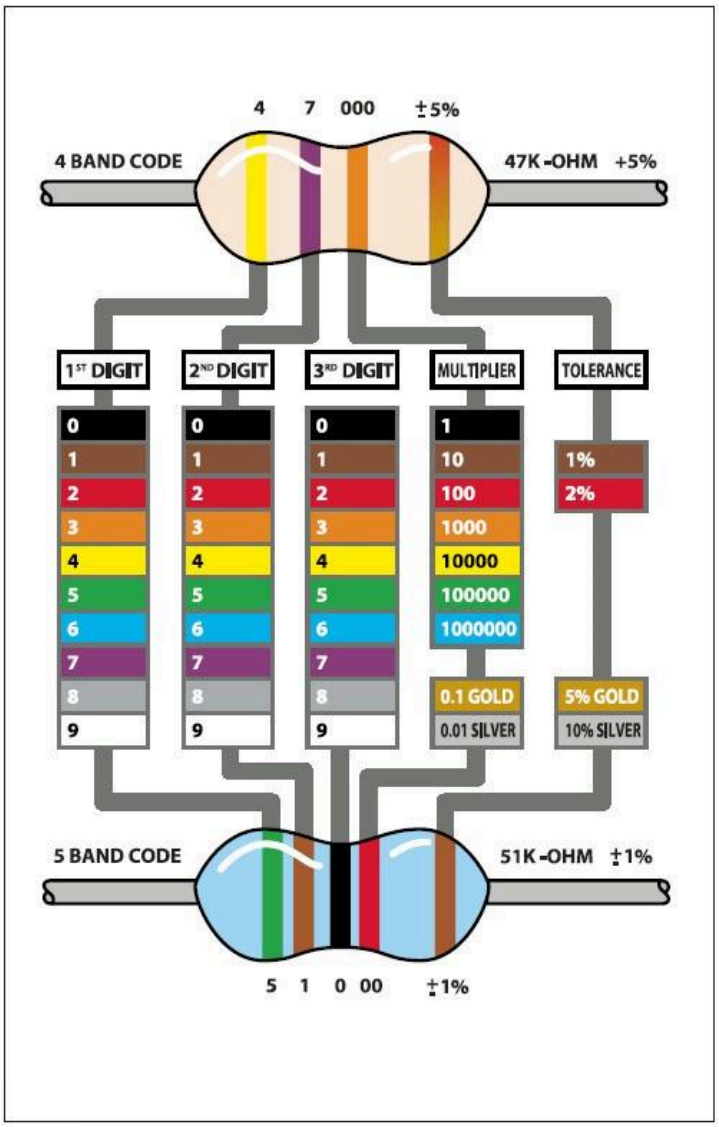


Arduino + LED + 220 Ω-resistor på breadboard. Lägg märke till att 1 kΩ och 10 kΩ som syns i bilden bara är alternativa värden — kursen använder 220 Ω.

Repetition av de viktigaste begreppen

RÄKNA BANDEN FÖRST

Kittet innehåller två sorters resistorer: **4-bands** (digit-digit-multiplikator-tolerans) och **5-bands** (digit-digit-digit-multiplikator-tolerans, 1 % precision). Titta på **din** resistor och räkna bandens antal innan du börjar dekoda. 220 Ω är **röd-röd-brun-guld** (4-band) eller **röd-röd-svart-svart-brun** (5-band). Osäker? Mät med multimeter.



Färgkodschart. Överst: 4-band (första två siffror, multiplikator, tolerans). Underst: 5-band (tre siffror, multiplikator, tolerans). Samma tabell sitter inuti locket på ditt kit.

KRETSEN

Du har fyra saker på breadboarden: Arduinon, en LED, en 220 Ω-resistor, och två kablar. Strömmen går ut från digital pin 13 när den är `HIGH`, in i LED:ens långa ben, ut genom korta benet, in i ena änden av resistorn, ut från andra änden, och tillbaka in i Arduinons GND-pinne. Det är en sluten ring.

Bryts ringen händer ingenting. En lös kabel, ett ben som sitter i fel hål, en resistor som glömts bort — alla tre är samma fel: ringen är inte sluten. Felsök genom att följa strömmen med fingret, hål för hål, från pin 13 tillbaka till GND.

> **VATTENANALOGIN**
> Om elektricitet känns abstrakt: tänk vatten. **Spänningen** (volt) är trycket i systemet — som hur högt en vattentank står över ditt kök. **Strömmen** (ampere) är flödet — hur

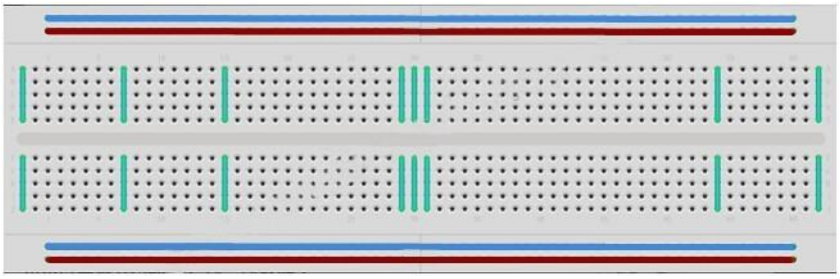
> mycket vatten som faktiskt rör sig per sekund. **Motståndet** (ohm) är hur mycket röret stryper flödet — som en halvöppen kran. En **resistor** är en strypventil.
>
> Ohms lag säger: för ett givet tryck bestämmer strypningen flödet. Stryp mer → mindre flöde. Lossa på → mer flöde. Vi kommer återvända till den här bilden flera gånger under kursen, bland annat när vi bygger spänningsdelaren i Modul 4.

BREADBOARDEN

Plastbiten du byggde på är ihålig med små metallklämmor invändigt. **Fem hål i rad är samma elektriska nod.** Raden bredvid är en helt annan nod. Gapet i mitten av brädan bryter förbindelsen mellan övre och nedre halvan.

Längs sidorna löper två par långa rälar: + och −. Dessa **power rails** kommer bli viktiga först i Modul 2, men tänk på dem som ”ett långt hål vardera” redan nu.

När LED:en inte lyser: kolla först att kabeln och komponentbenet faktiskt sitter på **samma rad**. Nio av tio nybörjarfel är ett hål fel.



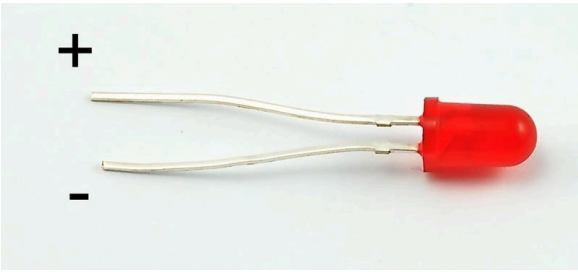
Breadboarden ovanifrån, med röda och blåa power rails längs sidorna och de gröna streck som visar vertikala nod-kolumner i mittsektionen. Gapet i mitten bryter förbindelsen mellan övre och nedre halvan.

OHMS LAG

Tre storheter, en formel:

$$U = I \cdot R$$

Omformulerat: $I = U/R$, $R = U/I$. Håll för den storhet du vill räkna ut, så ser du vilka de andra två ska göras av. Fullständig förklaring med exempel finns i Bilaga F.



LED:ens polaritet — långt ben (anod) till plus, kort ben med platt kant (katod) till GND.

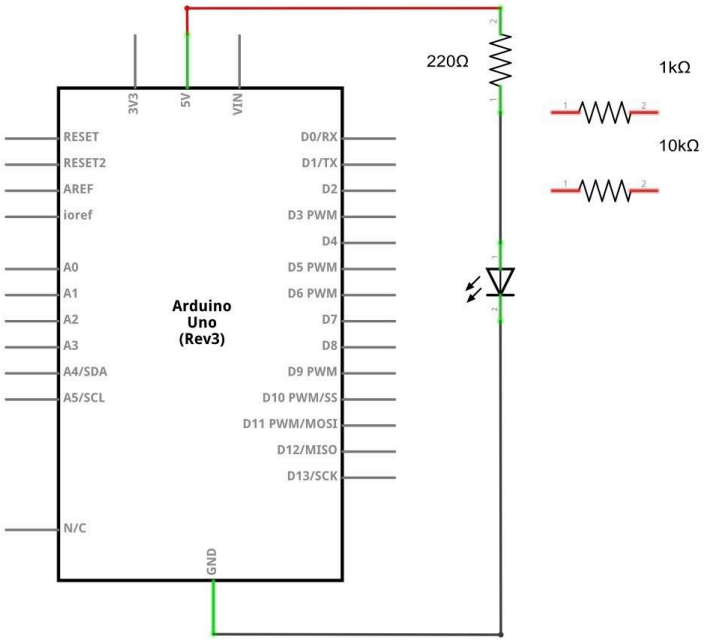
Resistorn läses från den ände där färgbanden ligger tätast. 220 Ω = röd-röd-brun-guld (4-band) eller röd-röd-svart-svart-brun (5-band). Bilaga C har full tabell.

VARFÖR 220 Ω?

En vanlig röd LED har ett **framspänningsfall** på ungefär 2 V. Det betyder att när du skickar 5 V över kombinationen LED + resistor, så ”äter LED:en upp” 2 V och resistorn måste ta hand om resten. Vi siktar på en ström runt 15 mA genom LED:en — ett säkert riktvärde långt under datablads-maxvärdet på 20 mA, men tillräckligt för att lysa synligt:

$$R = (5\ \text{V} - 2\ \text{V}) / 15\ \text{mA} = 200\Omega$$

220 Ω är det närmaste standardvärdet, och det ger en ström runt 13,6 mA — inom säkert område för varje LED i kittet. Djupare förklaring av varför LED:en inte följer Ohms lag direkt finns i Bilaga F.



Schematisk krets för Blink: Arduino Uno → D13 → LED (anod) → katod → 220 Ω → GND. Den här symboliska ritningen är densamma som breadboard-kopplingen — bara på ett annat språk.

`DIGITALWRITE`, `PINMODE` OCH `LED_BUILTIN`

`pinMode(LED_BUILTIN, OUTPUT)` talar om för Arduinon: ”den här pinnen är en utgång, jag ska styra den”. `digitalWrite(LED_BUILTIN, HIGH)` sätter pinnen till 5 V. `digitalWrite(LED_BUILTIN, LOW)` sätter den till 0 V. `delay(ms)` är en paus i antal millisekunder. Allt `Blink` gör är att växla mellan HIGH och LOW med pauser emellan.

`LED_BUILTIN` är Arduinons eget **alias** för pin 13 — exakt samma pinne, bara ett annat namn. När du skriver `digitalWrite(LED_BUILTIN, HIGH)` går samma ström ut genom pin 13-hålet som när du skriver `digitalWrite(13, HIGH)`. Varför två namn? Pin 13 är fysiskt kopplad till en liten ytmonterad LED på kortet (märkt ”L”) — permanent, via sitt eget motstånd fabriksvägen. Arduinons example-sketchar använder `LED_BUILTIN` så att samma kod fungerar

på alla deras kort, även de där inbyggda LED:en inte sitter på pin 13. Vi använder samma namn i kursen för att matcha Arduino IDE:s exempel.

När ni blinkar `LED_BUILTIN` blinkar alltså både den lilla inbyggda LED:en på kortet **och** er externa LED på breadboarden — båda är kopplade till samma pin. Den externa LED:en bygger ni för att se en krets ni själva monterat.

> **OM UPLOAD INTE GÅR IGENOM**
> – Tools → Port → välj den port som dyker upp när ni stoppar in USB-kabeln (`/dev/cu.usbmodem*` på Mac, `COM*` på Windows).
> – Tools → Board → ”Arduino Uno”.
> – Byt USB-kabel om porten inte dyker upp alls — vissa billiga kablar saknar dataledare.
>
> Fullständig felsökning i Bilaga B, avsnitt ”Upload-fel”.

Bygg från minnet

Lägg undan slidesen. Ta fram Arduinon, en LED, en resistor och två kablar. Försök koppla blinkkretsen utan att titta på bild. När lampan blinkar: grattis — du kan det.

Fastnar du, peka ut dessa fyra frågor:

1. Vilken pinne ska ström gå **ut** ur?
2. Vilket ben på LED:en ska strömmen in i?
3. Var i ringen ska resistorn sitta?
4. Vilket ben på Arduinon ska strömmen tillbaka till?

Svaren: pin 13, långa benet, i serie någonstans i ringen, GND.

Hemma-övningar

ÖVNING 1 — HJÄRTSLAG

Ändra `Blink` så att lampan blinkar i ett hjärtslagsmönster: två korta blink tätt efter varandra, sedan en längre paus, och om igen.

> **LEDTRÅD**
> Två `digitalWrite(HIGH)` / `digitalWrite(LOW)`-par med korta `delay()` emellan, och sedan ett längre `delay()` innan loopen börjar om.

ÖVNING 2 — NAMNGIVNA TIDER

Skriv om din hjärtslags-kod så att alla tidsvärden ligger i `const int`-variabler i toppen av filen. Ändra sedan bara variablerna för att testa ett snabbare tempo. Syftet: du ska aldrig behöva leta upp fem `delay`-siffror och ändra dem var för sig.

> **EXEMPEL**
> `const int kort = 80;` `const int lang = 200;` `const int paus = 1000;` — och sedan använda variabelnamnen i stället för siffror inne i `loop()`.

ÖVNING 2½ — MAMMA-IGENKÄNNING

Innan morse — bygg en övergång. Skriv ett mönster som är tydligt igenkännbart från vanlig Blink: **tre snabba blink** (100 ms på, 100 ms av) följt av en **lång paus** (1500 ms). Använd de `const int`-variabler du skapat i övning 2.

Detta kluster-av-blink är faktiskt morse-S — och byggstenen i nästa övning. Om du kan kluster-av-tre kan du morsa vilken bokstav som helst.

ÖVNING 3 — DINA INITIALER I MORSE

Slå upp morsealfabetet och få lampan att blinka dina initialer. `·` = en kort blinkning, `–` = en lång blinkning (tre gånger så lång som en kort). Mellan två bokstäver: en kort paus. I slutet av hela meddelandet: en längre paus innan det börjar om.

> **BÖRJA ENKELT**
> Du behöver bara `digitalWrite` och `delay` — ingen ny syntax. Skriv ut varje dit och dah som ett par: `digitalWrite(LED_BUILTIN, HIGH)`, sedan `delay(...)`, sedan `digitalWrite(LED_BUILTIN, LOW)`, sedan `delay(...)`. Bokstaven ”S” blir tre korta blink i följd. Det blir många rader kod — det är OK, du lär dig att upprepa. Om du senare vill rensa upp det, introducerar vi **funktioner** i Bilaga A — en funktion är ett namngivet kodblock man kan anropa om och om igen.

Vanliga fel och snabblösningar

LED lyser inte	Vänd LED:en — långt ben mot pin 13, kort mot resistorn.
LED lyser väldigt svagt	Kontrollera resistor-värdet. 220 Ω = röd-röd-brun-guld (4-band) eller röd-röd-svart-svart-brun (5-band). Om du dekodat fel kan du fått 2,2 kΩ (10× för hög) — dubbla multiplikatorn. Mät med multimeter vid tvivel.
`Blink` kompilerar men inget händer	Kabeln till pin 13 sitter i pin 12 eller GND (hålet precis bredvid pin 13). Räkna hålen.
`expected ';' before...`	Glömt semikolon på föregående rad. Arduino-IDE:ns felpanel pe-

	kar på nästa rad, felet är på raden ovanför.
`'digitalWrit' was not declared in this scope`	Stavfel — funktionen heter `digitalWrite`, du missar `e` på slutet. Arduino är också skiftlägeskänsligt, så `digitalwrite` (litet w) är också fel.
Upload misslyckas med `avrdude: stk500_getsync()`	Välj rätt port under Verktyg → Port. USB-kabeln (Uno använder Type B, den kvadratiska kontakten) måste vara en **datakabel**, inte en ren laddkabel.

Snabbreferens

`pinMode(pin, OUTPUT)`	Sätter upp pinnen som utgång. Görs en gång, i `setup()`.
`digitalWrite(pin, HIGH)`	Slår på pinnen (5 V). Strömmen går ut.
`digitalWrite(pin, LOW)`	Slår av pinnen (0 V). Ingen ström.
`delay(ms)`	Paus i millisekunder. `delay(1000)` = en sekund.
`const int name = 13;`	Ger ett tal ett namn. Används för pin-nummer och andra värden som inte ändras.
`LED_BUILTIN`	Arduinons inbyggda namn på pin 13. En liten ytmonterad LED (med eget seriemotstånd på kortet) är kopplad dit fabriksvägen — men den gäller **bara** den. Kopplar du en egen LED i header-hålet för pin 13 måste du fortfarande ha ett eget 220 Ω-motstånd i serie.

Inför nästa träff

Idag har Arduinon bara **pratat** till omvärlden — skickat ström ut genom en pinne. Det kallas **act** i embedded-världen. Från och med Modul 3 lär vi den också **lyssna** — läsa tillstånd in från knappar och sensorer. Mönstret **sense → act** — ”läs av → agera” — är grunden för allt ni kommer att bygga. Tänk på det hädanefter.

Men innan sense: en ny utgångs-teknik. Nästa vecka släpper vi `digitalWrite` i två minuter och lär oss `analogWrite` — ett kommando som inte bara kan sätta pinnen PÅ eller AV, utan kan ställa den någonstans **däremellan**. Det är det som gör det möjligt att blanda färg på en RGB-LED.

Ingen ny matematik, men en ny idé: Arduinon kan inte göra analog ström på riktigt — den **fuskar**, genom att blinka pinnen jättesnabbt. Hur snabbt? Det ska vi titta på.

Glöm inte dator eller kitet hemma!