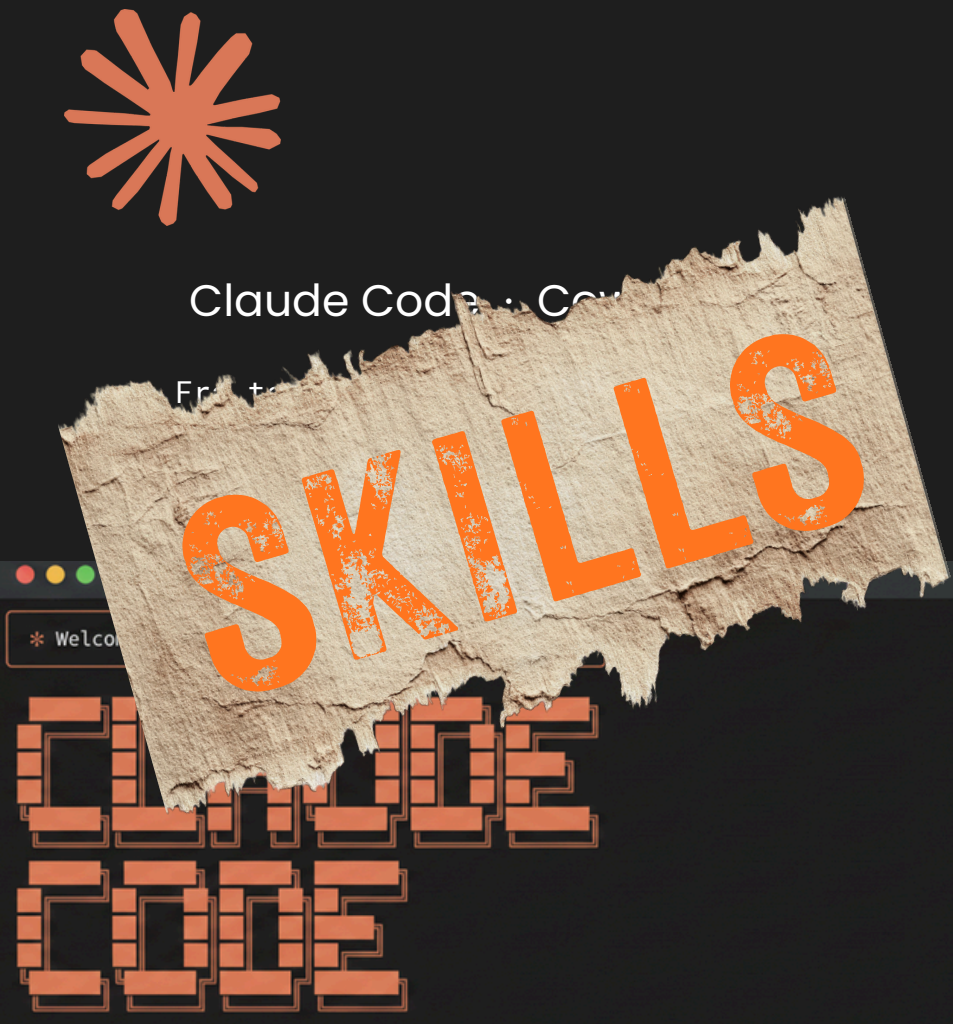
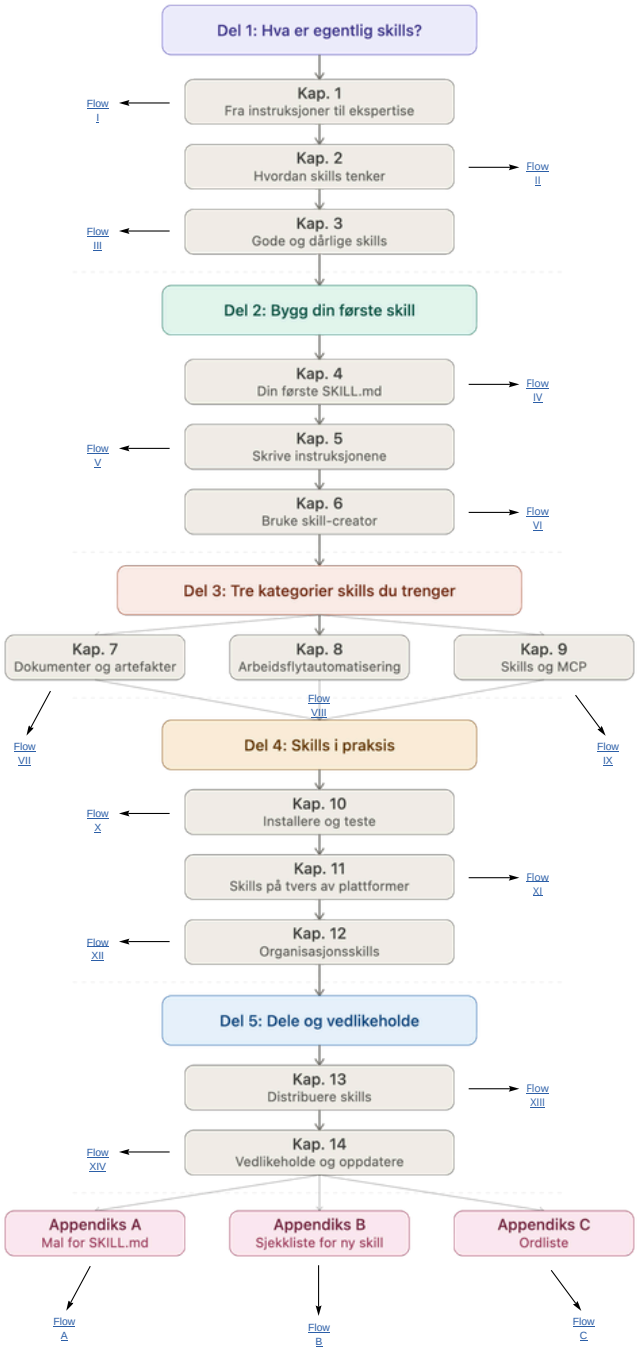



Kenneth Bareksten

# Flowchart


Del 1: Hva er egentlig skills?
Kap. 1
Fra instruksjoner til ekspertise
Flow I
Kap. 2
Hvordan skills tenker
Flow II
Kap. 3
Gode og dårlige skills
Flow III
Del 2: Bygg din første skill
Kap. 4
Din første SKILL.md
Flow IV
Kap. 5
Skrive instruksjonene
Flow V
Kap. 6
Bruke skill-creator
Flow VI
Del 3: Tre kategorier skills du trenger
Kap. 7
Dokumenter og artefakter
Kap. 8
Arbeidsflytautomatisering
Kap. 9
Skills og MCP
Flow VII
Flow VIII
Flow IX
Del 4: Skills i praksis
Kap. 10
Installere og teste
Flow X
Kap. 11
Skills på tvers av plattformer
Flow XI
Kap. 12
Organisasjonsskills
Flow XII
Del 5: Dele og vedlikeholde
Kap. 13
Distribuere skills
Flow XIII
Kap. 14
Vedlikeholde og oppdatere
Flow XIV
Appendiks A
Mal for SKILL.md
Appendiks B
Sjekkliste for ny skill
Appendiks C
Ordliste
Flow A
Flow B
Flow C

# Innledning

## Claude Code Skills

I forrige bok, Claude Code CLI, lærte du det grunnleggende: hvordan du installerer Claude Code, snakker med terminalen, bruker kommandoer og gir Claude oppgaver som faktisk gjennomføres. Du fikk Claude til å handle, ikke bare svare.

Denne boken tar deg et steg videre. Ikke et teknisk steg. Et strategisk steg.

Spørsmålet vi skal svare på er dette: Hva skjer når Claude ikke lenger trenger å spørre deg om alt? Når den allerede vet hvordan du vil ha ting gjort, hva slags tone du bruker, hvilke filer som er viktige og hvilke steg som alltid er de samme?

Det er det skills handler om.

## Hva du hadde etter første bok

Du hadde en assistent som var flink, men som startet fra null hver gang. Du måtte fortelle Claude hva du ville, hvordan du ville ha det og i hvilken rekkefølge ting skulle gjøres. Claude var kapabel, men uvitende om konteksten din.

Skills er opplæringen. Du skriver ned arbeidsmetoden din én gang, og deretter kan Claude følge den automatisk i alle fremtidige samtaler og på tvers av alle prosjekter.

## Hva du får ut av denne boken

- Du forstår hva en skill egentlig er, og hva som skiller den fra en vanlig instruksjon
- Du kan lage din første egne skill fra bunnen
- Du vet hvilke typer oppgaver som egner seg for skills og hvilke som ikke gjør det
- Du kan distribuere skills til kolleger eller bruke andres ferdige skills
- Du har konkrete eksempler tilpasset hverdagen din

**Øvelse 1:**

1. Åpne terminalen (Mac: Cmd + Mellomrom, skriv Terminal. Windows: Windows-tasten, skriv PowerShell).
2. Skriv: claude --version og trykk Enter.
3. Du skal se et versjonsnummer. Hvis du får en feilmelding, se installasjonsveiledningen i første bok.

**Tips**: *Denne boken forutsetter at du har lest eller har erfaring med grunnleggende Claude Code. Du trenger ikke huske alt, men du bør kjenne til terminalen og vite hva en slash-kommando er.*

# Fra instruksjon til ekspertise
## Kapittel 1

Tenk tilbake på siste gang du begynte en ny samtale med Claude. Du skrev hva du ville ha, kanskje forklarte du litt bakgrunn, og Claude hjalp deg. Neste gang startet du på nytt.

Det er ikke noe galt med det. Men det betyr at du bærer all konteksten. Du er den som husker hvordan ting skal gjøres, hva som er viktig og hva Claude bør unngå. Claude er flink, men den er som en blank tavle ved starten av hver økt.

## Hva en skill egentlig er

En skill er en mappe med instruksjoner. Ikke en samtale. Ikke en melding du skriver på nytt hver gang. En mappe på datamaskinen din som Claude leser når den er relevant for oppgaven du holder på med.

> **Analogi**: *Tenk på det som en opplæringsguide du skriver til en ny medarbeider. Du trenger ikke sitte ved siden av dem og forklare alt. Du har skrevet det ned én gang, og de slår opp når de trenger det.*

En skill skiller seg fra en vanlig instruksjon på én avgjørende måte: den aktiveres automatisk. Du skriver ikke en kommando for å starte den. Claude ser hva du holder på med, finner den relevante skillen, og bruker den uten at du trenger å be om det.

## Kommandoer og skills er ikke det samme

Du kjenner kanskje til slash-kommandoer fra forrige bok, Claude Code CLI. /help, /clear, /status. De starter du selv ved å skrive dem. Du velger verktøyet.

En skill er annerledes. Den er modellstyrt, ikke brukerstyrt. Det vil si at det er Claude som bestemmer at den er relevant, ikke du. Hvis du ber Claude skrive et Word-dokument, aktiverer den automatisk Word-skillen. Hvis du ber den lage en presentasjon, bruker den presentasjonsskillen. Det eneste du trenger å vite er at de finnes.

## Tre lag og derfor fungerer det

Det smarteste med skills er ikke at de finnes. Det er hvordan de er bygget opp. En skill lastes ikke inn i sin helhet på én gang. Den avslører seg i tre lag, og bare det som er nødvendig hentes inn.

**Lag 1: Beskrivelsen**

Øverst i skillfilen ligger en kort beskrivelse. Denne er alltid tilstede i Claudes system, som en liten lapp på oppslagstavlen. Den forteller Claude: «Jeg finnes, og dette er når du bør bruke meg.»

Selve innholdet i skillen lastes ikke inn ennå. Bare beskrivelsen. Det sparer plass og gjør Claude raskere.

**Lag 2: Instruksjonene**

Når Claude ser at skillen er relevant, leser den hoveddokumentet. Her ligger de faktiske instruksjonene, steg for steg. Claude bruker dette for å forstå hva den skal gjøre og i hvilken rekkefølge.

**Lag 3: Støttefiler**

Noen skills har ekstra filer: maler, skript, eksempler og stilguider. Disse leses bare inn hvis instruksjonene sier at de er nødvendige for akkurat denne oppgaven. Trenger Claude malen for månedlig rapport? Den lastes inn. Trenger den ikke det? Den forblir urørt.

> **Analogi**: *Det er som en ny ansatt som leser stilguiden bare når de skriver et brev, ikke mens de henter kaffe. Den riktige informasjonen, til rett tid.*

## Hva en skill-mappe inneholder

En skill er altså en mappe. Mappens innhold ser slik ut:

```
min-skill/
├── SKILL.md        ← Obligatorisk. Her ligger instruksjonene.
├── scripts/        ← Valgfritt. Kode Claude kan kjøre.
├── references/     ← Valgfritt. Ekstra dokumentasjon.
└── assets/         ← Valgfritt. Maler og ressurser.
```

SKILL.md er den eneste filen du alltid trenger. Alt annet er valgfritt, og du legger det til når oppgavene dine krever det.

**Toppen av SKILL.md: frontmatter**

Øverst i SKILL.md ligger det du kan tenke på som skiltkort. En liten blokk med strukturert informasjon som forteller Claude hvilken skill det er og når den skal brukes. Den ser slik ut:

```
---
name: ukentlig-rapport
  description: Genererer ukentlige statusrapporter i Word-format.
  Bruk når brukeren sier «lag rapport», «ukesoppsummering»
  eller «skriv status».
---
```

**Name** er det maskinen bruker for å identifisere skillen. **Description** er det Claude leser for å forstå om skillen er relevant akkurat nå.

> **Tips**: *Beskrivelsen er det viktigste du skriver i hele skillen. En god beskrivelse betyr at Claude bruker skillen når den skal. En dårlig beskrivelse betyr at den aldri aktiveres selv om instruksjonene inni er perfekte.*

En skill du lager i Claude Code fungerer på claude.ai i nettleseren og i API-en. Du lager den én gang, og den følger med. Det er mulig fordi Anthropic i desember 2025 publiserte skills som en åpen standard, og samme format brukes nå av flere AI-verktøy.

For deg som bruker betyr det at du ikke trenger å lage samme skill to ganger for to forskjellige steder du bruker Claude.

## Hvor skills lagres

Skills kan ligge to steder, avhengig av hva du vil:

- Personlige skills: ~/.claude/skills/ — tilgjengelig i alle prosjekter på maskinen din
- Prosjektskills: .claude/skills/ i rotmappen til et prosjekt — kun tilgjengelig i det prosjektet

Hvis du lager en skill for å skrive tilbakemeldinger i en bestemt tone, legger du den gjerne som en personlig skill. Hvis du lager en skill for et spesifikt skolefag med egne maler, er det mer fornuftig å legge den i prosjektmappen for det faget eller prosjektet.

## En skill er ikke magi, det er metodikk

Det er lett å tenke på skills som noe avansert og teknisk. Det er de ikke. En skill er grunnleggende sett en nedskrevet arbeidsmetode. Det du gjerne har i hodet, eller som du forklarer til en ny kollega, eller som du skriver i en sjekkliste. Det er en skill.

Det tekniske — SKILL.md, frontmatter, mappestruktur er bare formen den tar for at Claude skal forstå den.

Du har sikkert allerede skills i hodet ditt. Rutiner du følger. Ting du alltid gjør i en bestemt rekkefølge. Måter du vil at tekster skal høres ut. Å lage en skill er å skrive det ned på en måte Claude også kan bruke.

**Øvelse 2:**

1. Tenk på noe du gjør jevnlig med Claude. Et tilbakevendende skrivearbeid, en rapport, en type dokument.
2. Skriv ned tre til fem steg som alltid er de samme, uansett hvilken konkret versjon av oppgaven du jobber med.
3. Lagre dette midlertidig i et tekstdokument. Det er råmaterialet til din første skill.

## Husk dette



- En skill er en mappe med instruksjoner Claude aktiverer automatisk når de er relevante.
- Skills er modellstyrte, ikke brukerstyrte. Forskjellig fra slash-kommandoer.
- Tre lag sikrer at bare nødvendig informasjon lastes inn, dette gjør skills effektive med mange instruksjoner.
- SKILL.md er den eneste obligatoriske filen. Frontmatter-beskrivelsen er det viktigste du skriver.
- Skills fungerer på tvers av Claude Code, claude.ai og API. Lag én gang, bruk overalt.

# Hvordan en skills “tenker”
# Kapittel 2

I kapittel 1 lærte du at en skill er bygget opp i tre lag. Nå skal vi se nærmere på hva som faktisk skjer når Claude møter en oppgave og bestemmer seg for å bruke en skill. Det er ikke magi. Det er en veldig praktisk beslutning.

## Slik “tenker” skills

Se for deg at du åpner Claude Code og skriver: «Lag et sammendrag av denne rapporten og lagre det som Word-dokument.»

Claude leser meldingen og skanner skill-beskrivelsene den har tilgjengelig. Den ser: «Word-dokument-skill — bruk når bruker ber om .docx, rapport eller tekstdokument.» Det matcher. Skillen aktiveres.

Nå skjer de tre lagene i rekkefølge:

| Lag | Hva lastes | Hvorfor bare dette? |
|---|---|---|
| **1** | Beskrivelsen (YAML) | Claude må alltid vite hvilke skills som finnes, men trenger ikke innholdet ennå. |
| **2** | SKILL.md-instruksjonene | Nå vet Claude at denne skillen er relevant og trenger de faktiske instruksjonene. |
| **3** | Støttefiler (ved behov) | Instruksjonene refererer til en mal. Bare da hentes malen inn. |

> **Tips:** *YAML er et enkelt tekstformat for å skrive strukturert informasjon. Du trenger ikke kunne det, du trenger bare å vite at de tre strekene øverst i SKILL.md (---) markerer starten og slutten på denne blokken, og at alt inni er navn og verdier. Det ser ut som: navn: verdi. Mer avansert enn det blir det sjelden*

Resultatet: Claude bruker bare det den trenger, akkurat når den trenger det. Har du ti skills installert, lastes ikke alle ti inn samtidig. Bare de som er relevante for akkurat denne oppgaven.

Du kan sjekke om en skill ble aktivert ved å se på Claudes tankekjede. Navnet på skillen dukker opp når den leses inn.

## En åpen standard og hva det betyr for deg

Anthropic lanserte skills i oktober 2025. I desembet ble skills-formatet en åpen standard. Det betyr at formatet ikke lenger tilhører ett selskap, det er en felles spesifikasjon, litt som HTML er en åpen standard for nettsider.

Resultatet: den samme SKILL.md-filen du lager for Claude Code fungerer nå i flere AI-verktøy. Du trenger ikke lage skill-en på nytt for hvert verktøy du bruker. En skill du lager i dag er ikke bundet til Claude. Den er bundet til arbeidsmetoden din. Og den reiser med deg.

For deg som bruker betyr det konkret tre ting:

- **Portabilitet**: Skills du lager i Claude Code fungerer også på claude.ai og via API, uten endringer.
- **Gjenbruk**: Andres skills som følger standarden kan du bruke direkte, uavhengig av hvem som lagde dem.
- **Fremtidssikring**: Arbeidet du legger ned i å lage gode skills er ikke bortkastet om du bytter verktøy.

# Økosystemet: skills du ikke trenger å lage selv

Du trenger ikke starte fra bunnen. Det finnes et voksende utvalg av ferdige skills du kan hente og bruke med en gang.

**Anthropics egne skills**

Anthropic vedlikeholder et offisielt GitHub-repos med skills for de vanligste oppgavene: Word-dokumenter, Excel-regneark, PowerPoint-presentasjoner og PDF-behandling. Disse er de samme skillene som brukes under overflaten når du lager filer i claude.ai.

```
https://github.com/anthropics/skills
```

**Fellesskapet**

På agentskills.io og i GitHub-repoet awesome-claude-skills finner du skills laget av andre brukere. Kategoriene spenner fra frontend-design og kodegjennomgang til markedsføring, databehandling og skriveflyt.

Før du lager noe fra bunnen, er det verdt å sjekke om noen allerede har løst det samme behovet.

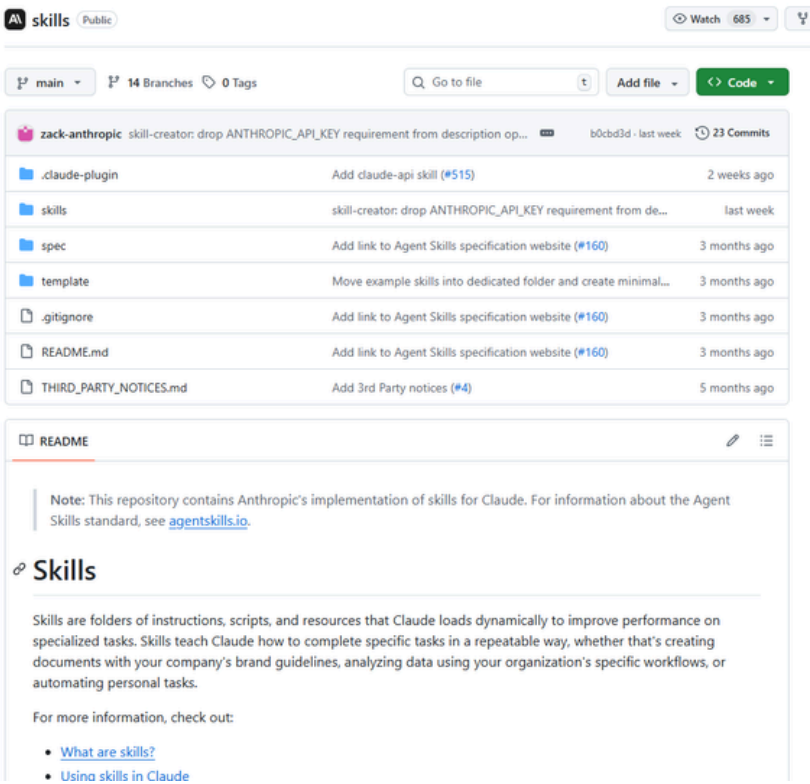
**Skill-creator**

Anthropic har laget en skill for å lage skills. Den heter skill-creator og er tilgjengelig i det offisielle repoet. Du aktiverer den, forteller Claude hva du vil automatisere og den hjelper deg gjennom prosessen steg for steg. Vi bruker den i kapittel 6.

## Husk dette



- Progressiv avsløring betyr at Claude bare laster inn det den faktisk trenger — beskrivelse, instruksjoner og støttefiler i tur og orden.
- Skills er siden desember 2025 en åpen standard. Én skill fungerer på tvers av Claude Code, claude.ai og API.
- Det finnes et rikt økosystem av ferdige skills. Sjekk GitHub og agentskills.io før du bygger noe fra bunnen.
- Skill-creator er en skill for å lage skills. Vi bruker den i kapittel 6.




Bilde: Et utdrag fra Github-repoet anthropic/skills/public

# Gode og dårlige skills
## Kapittel 3

Du vet nå hva en skill er og hvordan den aktiveres. Det neste spørsmålet er det viktigste: hva gjør at en skill faktisk fungerer i praksis? For en skill kan være teknisk korrekt bygget og likevel aldri brukes eller brukes på feil tidspunkt.

Det handler nesten alltid om én ting: beskrivelsen.

## Beskrivelsen bestemmer alt

Når Claude mottar en oppgave, skanner den gjennom YAML-beskrivelsene til alle skills den har tilgjengelig. Den stiller seg spørsmålet: er noen av disse relevante for det brukeren vil nå?

Hvis beskrivelsen er upresis, vag eller for bred, er svaret nei selv om selve instruksjonene inne i skillen er gode. Skillen forblir ubrukt.

Se på forskjellen:

| Dårlig beskrivelse | God beskrivelse |
|---|---|
| Hjelper med dokumenter og skriving.<br><br>For generell til å matche noe spesifikt. Claude vet ikke når den skal brukes. | Lager ukentlige statusrapporter i Word-format. Bruk når bruker sier «lag rapport», «ukesoppsummering» eller «skriv status».<br><br>Konkret oppgave, konkrete triggere. Claude vet nøyaktig når den skal aktiveres. |
| AI-assistent for prosjektarbeid og planlegging.<br><br>For bred. Kan aktiveres på nesten hva som helst – og dermed stole på ingenting. | Lager prosjektplaner med milepæler og ansvarsoversikt. Bruk når bruker nevner «prosjektplan», «milepæler» eller «hvem gjør hva».<br><br>Avgrenset til én oppgave med tydelige triggere. |

Formelen er enkel: én konkret oppgave pluss to til fire setninger om når skillen skal brukes. Ikke mer.

> **Tips:** *Test beskrivelsen din ved å spørre deg selv: hvis jeg ikke visste noe om denne skillen og leste bare disse linjene, ville jeg forstått nøyaktig hva den gjør og når? Hvis svaret er nei, skriv om.*

## Kontekstbudsjett

Claude har et begrenset arbeidsminne som kalles kontekstvinduet. Alle skill-beskrivelser lastes automatisk inn i dette vinduet ved oppstart, siden Claude trenger dem for å vite hva som er tilgjengelig.

Det er satt av plass til skill-beskrivelser tilsvarende 2 prosent av det totale kontekstvinduet. Det høres mye ut, men hvis du har mange skills med lange beskrivelser, fyller de opp.

Konsekvensen er at skills med lavest relevans kan bli kuttet ut av oversikten. Claude varsler om dette hvis det skjer og du kan sjekke med kommandoen /context i terminalen.

```
/context
```

Løsningen er ikke å ha færre skills, men å holde beskrivelsene korte og presise. En god beskrivelse er sjelden mer enn to til tre setninger.

## Oppgaver som ikke passer til skills

En skill er ikke riktig verktøy for alt. Noen oppgaver egner seg bedre som vanlige instruksjoner i chatten, eller som enkle slash-kommandoer.

- Engangsoppgaver. Skal du bare gjøre noe én gang, er det raskere å skrive det i chatten enn å bygge en skill.
- Oppgaver som endrer seg ofte. Hvis arbeidsmetoden din er i konstant endring, vil en skill raskt bli utdatert og gjøre mer skade enn nytte.
- Sensitive opplysninger. Ikke legg personnumre, passord eller konfidensielle data inn i skills. Skills er filer på maskinen din, de bør behandles som det.
- For brede arbeidsflyter. En skill som prøver å dekke alt du gjør på jobb vil aldri aktiveres på riktig tidspunkt. Del den heller i flere, avgrensede skills.

> **Tips:** *En god skill gjør én ting godt. En dårlig skill prøver å gjøre alt og ender opp med ingenting.*

## Hva som kjennetegner en god skill

Oppsummert er dette tegnene på at en skill er bygget riktig:

- Beskrivelsen inneholder én konkret oppgave og minst to eksempler på triggere.
- Skillen aktiveres konsekvent på relevante oppgaver og ikke på irrelevante.
- Instruksjonene inni SKILL.md er steg-for-steg og ikke åpne for tolkning.
- Støttefilene er referert til fra SKILL.md, ikke bare lagt ved uten kontekst.
- Beskrivelsen er under 200 tegn.

Det siste punktet er en god tommelfingerregel. Klarer du ikke å beskrive hva skillen gjør på under 200 tegn, er den sannsynligvis for bred.

**Øvelse 3:**

1. Ta råmaterialet du skrev i øvelse 2, de tre til fem stegene som alltid er de samme.
2. Skriv en beskrivelse på maks to setninger: hva gjør skillen, og to eksempler på hva brukeren kan si for å utløse den.
3. Tell tegnene. Er du under 200? Er oppgaven avgrenset nok til at du ville forstått den uten forkunnskaper?

## Husk dette



- Beskrivelsen er det viktigste du skriver. Upresis beskrivelse betyr at skillen aldri aktiveres.
- En god beskrivelse inneholder én konkret oppgave og eksempler på triggere. Maks 200 tegn.
- Kontekstbudsjettet er begrenset. Korte, presise beskrivelser er bedre enn lange og vage.
- Skills egner seg ikke for engangsoppgaver, arbeidsflyter i konstant endring eller sensitive data.
- En skill som prøver å gjøre alt gjør ingenting. Del heller opp i flere avgrensede skills.

| name | description | license |
|---|---|---|
| frontend-design | Create distinctive, production-grade frontend interfaces with high design quality. Use this skill when the user asks to build web components, pages, artifacts, posters, or applications (examples include websites, landing pages, dashboards, React components, HTML/CSS layouts, or when styling/beautifying any web UI). Generates creative, polished code and UI design that avoids generic AI aesthetics. | Complete terms in LICENSE.txt |

This skill guides creation of distinctive, production-grade frontend interfaces that avoid generic "AI slop" aesthetics. Implement real working code with exceptional attention to aesthetic details and creative choices.

The user provides frontend requirements: a component, page, application, or interface to build. They may include context about the purpose, audience, or technical constraints.

### Design Thinking

Before coding, understand the context and commit to a BOLD aesthetic direction:

- **Purpose:** What problem does this interface solve? Who uses it?
- **Tone:** Pick an extreme: brutally minimal, maximalist chaos, retro-futuristic, organic/natural, luxury/refined, playful/toy-like, editorial/magazine, brutalist/raw, art deco/geometric, soft/pastel, industrial/utilitarian, etc. There are so many flavors to choose from. Use these for inspiration but design one that is true to the aesthetic direction.
- **Constraints:** Technical requirements (framework, performance, accessibility).
- **Differentiation:** What makes this UNFORGETTABLE? What's the one thing someone will remember?

**CRITICAL:** Choose a clear conceptual direction and execute it with precision. Bold maximalism and refined minimalism both work - the key is intentionality, not intensity.

Then implement working code (HTML/CSS/JS, React, Vue, etc.) that is:

- Production-grade and functional
- Visually striking and memorable

Bilde: Et utdrag fra skillen frontend-design på Github-repoet anthropic/skills/public

# Din første SKILL.md
# Kapittel 4

Fram til nå har det handlet om hva skills er og hvorfor de fungerer. Nå bygger vi. I dette kapittelet lager du din første komplette SKILL.md fra tom fil til noe Claude faktisk kan bruke

## Navneregler du ikke kan ignorere

Før du skriver en eneste linje, er det noen regler som må sitte. De er tekniske, men enkle:

- Filnavnet må være nøyaktig SKILL.md, med store bokstaver, ingen varianter.
- Mappens navn skal skrives med små bokstaver og bindestrek mellom ord. Dette kalles kebab-case, fordi ordene henger sammen med bindestreker, akkurat som et kjøttstykker på er kebab-spyd. Ukentlig-rapport er riktig. Ukentlig Rapport (mellomrom) og ukentlig_rapport (understrek) er feil.
- Ingen README.md inne i skill-mappen. All dokumentasjon hører hjemme i SKILL.md eller i references/-mappen.
- Ikke bruk claude eller anthropic i mappenavnet. Disse er reserverte.

> **Tips:** *Disse reglene er ikke anbefalinger. Bryt dem og Claude finner ikke skillen din.*

## Frontmatter-feltene

Øverst i SKILL.md ligger frontmatter-blokken. Du kjenner den igjen på de tre strekene. Her er alle feltene:

```
---
name: skill-navn # Obligatorisk. Kebab-case.
description: Hva den gjør. # Obligatorisk. Maks 1024 tegn.
 Bruk når bruker sier... # Inkluder triggere her.
 license: MIT # Valgfritt. Apache-2.0, MIT e.l.
compatibility: Claude Code # Valgfritt. Miljøkrav.
metadata: # Valgfritt. Egendefinerte felt.
 author: Ditt navn
 version: 1.0.0
---
```

De to obligatoriske feltene er **name** og **description**. Resten er valgfritt, men metadata med author og version er god vane hvis du planlegger å dele skillen.

**name**
Skal matche mappenavnet. Kebab-case, ingen mellomrom, ingen store bokstaver. Dette er maskinnavnet, ikke det du ser i grensesnittet, men det Claude bruker internt.

**description**

Dette er triggeren. Skriv hva skillen gjør og når den skal brukes. Inkluder konkrete fraser brukeren kan si. Husk: under 200 tegn for å være trygg, absolutt maks 1024.
compatibility

Nyttig hvis skillen krever noe spesielt — for eksempel at den bare fungerer i Claude Code og ikke i nettleseren, eller at den trenger nettverkstilgang. Ikke nødvendig for enkle skills.

**metadata**

Fritt felt. Author og version er de vanligste. Hvis du deler skillen på GitHub, hjelper det andre å forstå hvem som lagde den og hvilken versjon de ser på.

## Sikkerhetsregler

To regler som ikke er forhandlingsbare:

- Ingen XML-tegn (< og >) i frontmatter. Frontmatter lastes direkte inn i Claudes systemprompt. Spesialtegn kan ødelegge formateringen eller i verste fall injisere uønskede instruksjoner.
- Ikke legg sensitive data i skills. Passord, API-nøkler og personopplysninger hører ikke hjemme i en skill-fil. Skills er vanlige tekstfiler på maskinen din.

**En komplett SKILL.md**

Her er en ferdig SKILL.md du kan kopiere og tilpasse. Den følger alle reglene og har en beskrivelse som faktisk fungerer. NB! norske tegn som ø og æ bør unngås i mappenavn og tekniske identifikatorer, de kan skape problemer på tvers av operativsystemer.

```
---
name: motereferat
description: Skriver strukturerte møtereferater i Word-format med dato, deltakere,
beslutninger og oppfølgingspunkter.
 Bruk når bruker sier «lag referat», «møtenotat»,
 «skriv opp møtet» eller «referat fra i dag».'

metadata:
 author: Ditt navn
 version: 1.0.0
---
# Møtereferat
## Slik lager du referatet
1. Spør bruker om dato, deltakere og hva møtet handlet om
 hvis det ikke er oppgitt.
2. Lag et Word-dokument (.docx) med disse seksjonene:
 - Dato og tid
 - Deltakere
 - Agenda
 - Beslutninger (nummerert liste)
 - Oppfølgingspunkter med ansvarlig og frist
3. Bruk en nøytral, profesjonell tone.
4. Lagre filen som referat-YYYY-MM-DD.docx.
```

Legg merke til strukturen etter frontmatter: en overskrift, en nummerert arbeidsflyt, klare steg. Det er alt som trengs for en enkel skill.

**Øvelse 4:**

1. Opprett en ny mappe på maskinen din. Bruk et navn med små bokstaver og bindestrek, for eksempel moterefera uten norske tegn.

2. Lag en fil inni mappen som heter nøyaktig SKILL.md (store bokstaver, ingen varianter).

3. Skriv frontmatter-blokken: name settes til mappenavnet ditt, description hentes fra beskrivelsen du skrev i øvelse 2.

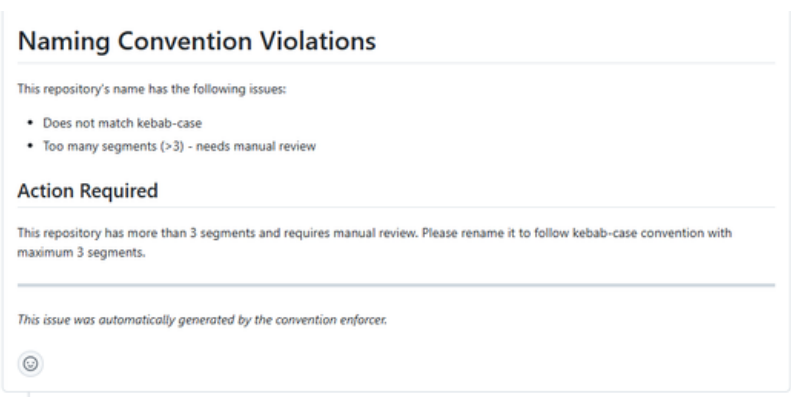
4. Under frontmatter: skriv en overskrift og instruksjonene som en nummerert liste. Bruk stegene fra øvelse 1.

5. Les gjennom som om du aldri hadde sett oppgaven før. Ville du forstått nøyaktig hva Claude skal gjøre?

## Husk dette



- Filnavnet må være nøyaktig SKILL.md. Mappens navn skal være kebab-case.
- name og description er de eneste obligatoriske feltene. Resten er valgfritt.
- Ingen XML-tegn i frontmatter. Ingen sensitive data i skills.
- Instruksjonene etter frontmatter skal være konkrete og steg-for-steg.
- Du har nå en fungerende skill. Neste kapittel viser hvordan du installerer og tester den.




Bilde: Feilmelding når kebab-case ikke blir opprettholdt.

# Skrive instruksjonene
# Kapittel 5

Frontmatter forteller Claude når skillen skal brukes. Instruksjonene forteller den hva den skal gjøre. Det er her det faktisk skjer og det er her mange gjør feil.

Den vanligste feilen er å skrive instruksjoner slik man ville skrevet dem til et menneske. Claude er ikke et menneske. Den trenger ikke motivasjon, kontekst eller bakgrunnshistorie. Den trenger presise, entydige steg.

## Tre instruksjonstyper

Velg type ut fra hva skillen skal gjøre:

**1. Nummererte steg**

Brukes når oppgaven har en fast rekkefølge. Claude følger stegene i orden og stopper ikke opp.

```
## Slik gjør du det

1. Spør bruker om dato og deltakere hvis ikke oppgitt.
2. Lag en overskrift med dato og møtetittel.
3. List opp deltakere.
4. Skriv beslutninger som nummerert liste.
5. Avslutt med oppfølgingspunkter med ansvarlig og frist.
6. Lagre som referat-YYYY-MM-DD.docx.
```

**2. Sjekkliste**

Brukes når rekkefølgen ikke er kritisk, men alle punkter må være med. Typisk for kvalitetskontroll eller gjennomgang.

```
## Sjekk før du leverer

- [ ] Tittel er under 70 tegn
- [ ] Ingress oppsummerer artikkelen i én setning
- [ ] Alle påstander har kilde eller er allment kjent
- [ ] Ingen setninger er over 25 ord
- [ ] Avslutningen har en klar konklusjon
```

**3. Beslutningstre**

Brukes når Claude må velge mellom ulike veier avhengig av situasjonen.

```
## Velg format

Hvis bruker ber om kort versjon:
 - Skriv maks 150 ord, ingen overskrifter

Hvis bruker ber om full versjon:
 - Bruk overskrifter, kulepunkter og eksempler

Hvis format ikke er spesifisert:
 - Spør: «Vil du ha kort eller full versjon?»
```

## Hva Claude faktisk følger

Claude er god på å følge instruksjoner som er:

- Konkrete. «Skriv tre kulepunkter» er bedre enn «oppsummer».
- Entydige. «Lagre som .docx» er bedre enn «lag et dokument».
- Avgrenset i lengde. «Maks 200 ord» er bedre enn «hold det kort».
- Eksplisitt om hva som mangler. «Spør bruker om dato hvis ikke oppgitt» er bedre enn å anta at Claude skjønner det selv.

Claude er dårlig på instruksjoner som er:

- Vage. «Skriv en god tekst» gir ingenting å jobbe med.
- For lange. En SKILL.md på 2000 ord er for mye. Del opp eller flytt detaljene til references/.
- Motstridende. «Hold det kort» i én setning og «inkluder alle detaljer» i neste forvirrer.

> **Tips:** *Skriv instruksjonene dine som om de skal følges av noen som er ny på jobben, flink, men uten å lese mellom linjene.*

## Referer til støttefiler

Hvis skillen din trenger en mal, et eksempel eller et referansedokument, legg det i references/-mappen og pek til det fra SKILL.md. Claude leser bare filen når instruksjonene sier at den er nødvendig.

```
min-skill/
├── SKILL.md
└── references/
    └── rapportmal.md
```

I SKILL.md peker du til filen slik:

```
## Mal

Bruk malen i references/rapportmal.md som utgangspunkt.
Fyll inn innholdet fra brukerens input.
```

**Tips:** *Ikke legg malen direkte inn i SKILL.md. Da lastes den inn hver gang skillen aktiveres, uansett om den trengs eller ikke. references/-filer lastes bare ved behov.*

## Dårlig vs bedre: To eksempler

Se på forskjellen i praksis:

| Dårlig instruksjon | Bedre instruksjon |
|---|---|
| Lag en profesjonell og velstrukturert rapport som dekker alle viktige punkter på en god måte. | Lag en rapport med disse seksjonene: Sammendrag (maks 100 ord), Funn (nummerert liste), Anbefaling (1-3 setninger). Lagre som .docx. |
| Hjelp brukeren med teksten og gjør den bedre. | Les teksten. Rett grammatikkfeil. Kutt setninger over 25 ord i to. Behold forfatterens tone — ikke skriv om fra bunnen. |

**Øvelse 5:**

1. Åpne SKILL.md du laget i øvelse 4.
2. Les instruksjonsdelen. Er stegene nummererte? Er hvert steg konkret og entydig? Inneholder noen steg vage ord som «god», «profesjonell» eller «passende»?
3. Bytt ut minst to vage formuleringer med konkrete instruksjoner. Bruk eksemplene fra tabellen over som mal.
4. Hvis skillen trenger en mal eller et eksempel, lag en references/-mappe og legg filen der. Pek til den fra SKILL.md.
5. Les gjennom på nytt. Kunne noen fulgt disse instruksjonene uten å spørre om noe?

## Husk dette



- Velg instruksjonstype etter oppgaven: nummererte steg, sjekkliste eller beslutningstre.
- Konkret og entydig slår vag og generell hver gang.
- Lange instruksjoner hører hjemme i references/, ikke i SKILL.md.
- Pek eksplisitt til støttefiler, Claude henter dem ikke automatisk.

# Bruke SKILL-creator
## Kapittel 6

Du har nå laget en skill for hånd. Du kjenner strukturen, vet hva frontmatter gjør og forstår hva som skiller en god instruksjon fra en dårlig. Det betyr at du er klar til å bruke skill-creator og faktisk forstå hva du får ut av den.

Skill-creator er en skill som hjelper deg å lage andre skills. Anthropic har bygget den inn i det offisielle GitHub-repositoriet, og den fungerer ved at Claude stiller deg spørsmål, samler svarene og skriver SKILL.md for deg. Resultatet er ikke alltid perfekt, men det er et solid utgangspunkt.

## Installere SKILL-creator

Skill-creator ligger i Anthropics offisielle skills-repositorium på GitHub. Slik får du den:

```
# Klon repositoriet
git clone https://github.com/anthropics/skills.git

# Kopier skill-creator til din personlige skills-mappe
cp -r skills/skill-creator ~/.claude/skills/
```

Har du ikke git installert, kan du gå til github.com/anthropics/skills i nettleseren, laste ned skill-creator-mappen som zip-fil og pakke den ut i ~/.claude/skills/.

> **Tips:** *~/.claude/skills/ er mappen for personlige skills som gjelder alle prosjekter på maskinen din. Hvis mappen ikke finnes, lag den: mkdir -p ~/.claude/skills/*

## Bekreft at den er tilgjengelig

Start Claude Code og sjekk at skillen er lastet inn:

```
/context
```

Du skal se skill-creator i listen over tilgjengelige skills. Hvis den ikke dukker opp, sjekk at mappen ligger riktig og at filnavnet er nøyaktig SKILL.md.

## Slik bruker du skill-creator

Du trenger ikke kalle på skill-creator direkte. Fortell Claude at du vil lage en ny skill, og den aktiveres automatisk:

```
Jeg vil lage en skill for å generere ukentlige
prosjektoppdateringer i e-postformat.
```

Claude starter da en strukturert dialog. Den stiller spørsmål i denne rekkefølgen:

- Hva skal skillen gjøre? Beskriv oppgaven med egne ord.
- Hva sier brukeren for å utløse den? Claude vil ha konkrete eksempler på triggere.
- Hvilke steg skal Claude følge? Her henter du frem arbeidsflyten din.
- Trenger skillen støttefiler? Maler, eksempler, referanser.

Når du har svart, genererer Claude mappestrukturen og skriver SKILL.md. Du får se innholdet direkte i terminalen og kan si ja til å lagre den.

## Hva du bør sjekke etterpå

Skill-creator er et godt verktøy, men den vet ikke det du nå vet. Åpne den ferdige SKILL.md og gå gjennom disse punktene:

- Er beskrivelsen under 200 tegn? Skill-creator har en tendens til å skrive litt lange beskrivelser. Kutt det ned.
- Er triggerne konkrete nok? Sjekk at beskrivelsen inneholder eksempler på hva brukeren faktisk skriver, ikke bare hva oppgaven handler om.
- Er instruksjonene entydige? Gå gjennom hvert steg med blikket fra kapittel 5. Bytt ut vage formuleringer.
- Er skillen for bred? Hvis du svarte bredt på spørsmålene, kan resultatet bli en skill som prøver å gjøre for mye. Del den opp om nødvendig.

> **Tips:** *Skill-creator gir deg 80 prosent av jobben på 20 prosent av tiden. De siste 20 prosentene, som bestemmer om skillen faktisk fungerer, er din jobb.*

## Manuelt vs. skill-creator

Nå har du erfaring med begge metodene. Her er når du bør bruke hvilken:

| | Manuelt | creator |
|---|---|---|
| Enkel, avgrenset oppgave | ✓ | ✓ |
| Kompleks arbeidsflyt | ✓ | Delvis |
| Du vet nøyaktig hva du vil | ✓ | Fungerer |
| Du er usikker på strukturen | Tregere | ✓ |
| Du vil lære hvordan skills fungerer | ✓ | Ikke ideelt |

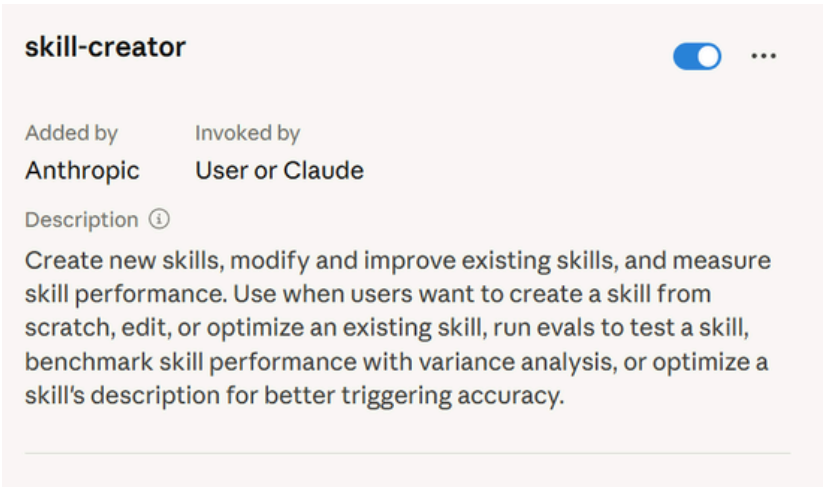
**Øvelse 6:**

1. Installer skill-creator i ~/.claude/skills/ som beskrevet over.
2. Start Claude Code og bekreft med /context at skill-creator er tilgjengelig.
3. Be Claude lage en skill for en annen oppgave enn den du lagde manuelt – noe du faktisk vil bruke.
4. Når SKILL.md er generert: åpne den og gå gjennom sjekklisten i dette kapittelet. Juster det som ikke holder.
5. Sammenlign den genererte SKILL.md med den du lagde manuelt. Hva er likt? Hva ville du gjort annerledes?

## Husk dette



- Skill-creator er en skill som lager andre skills, tilgjengelig fra github.com/anthropics/skills.
- Den stiller spørsmål og skriver SKILL.md for deg. Du må likevel gå gjennom resultatet.
- Sjekk alltid beskrivelseslengde, triggere og instruksjonspresisjon etter generering.
- Manuell bygging gir bedre forståelse. Skill-creator gir raskere utgangspunkt.




Bilde: skill-creator i claude.ai

# Dokumenter og artefakter
# Kapittel 7

Den første og mest brukte kategorien skills handler om å lage filer. Word-dokumenter, presentasjoner, regneark, PDF. Dette er oppgaver de fleste gjør jevnlig, og det er her skills gir umiddelbar verdi.

Det fine med denne kategorien er at du ikke trenger å starte fra bunnen. Anthropic har allerede laget skills for de vanligste dokumenttypene, og de er tilgjengelige gratis.

## Anthropics ferdige dokument-skills

Disse fire skillene er bygget og vedlikeholdt av Anthropic. De er de samme skillene som brukes under overflaten når du lager filer i claude.ai:

| Skill | Format | Aktiveres når du sier … |
|---|---|---|
| docx | Word (.docx) | lag dokument, skriv rapport, lag brev, Word-fil |
| pptx | PowerPoint (.pptx) | lag presentasjon, lag slides, PowerPoint |
| xlsx | Excel (.xlsx) | lag regneark, lag tabell, Excel-fil |
| pdf | PDF (.pdf) | lag PDF, fyll ut skjema, PDF-dokument |

Hent dem fra github.com/anthropics/skills og legg dem i ~/.claude/skills/ — samme fremgangsmåte som med skill-creator i kapittel 6.

## Hva som er spesielt med dokument-skills

Dokument-skills bruker Claudes innebygde evne til å lage filer. Skillen selv inneholder ikke kode for å generere et Word-dokument, den inneholder instruksjoner om hva dokumentet skal inneholde, hvordan det skal struktureres og hva Claude skal spørre om hvis noe mangler.

Det betyr at du kan lage en dokument-skill uten å kunne programmering. Alt du trenger er å beskrive hvordan dokumentet ditt skal se ut.

> **Tips:** *Anthropics docx-skill vet hvordan man lager et Word-dokument. Din skill vet hva det dokumentet skal inneholde. Sammen produserer de nøyaktig det du vil ha.*

## Lage din egen dokument-skill

Den virkelige verdien kommer når du kombinerer en Anthropic-skill med din egen. Se for deg at du alltid lager månedlige statusrapporter med samme struktur. I stedet for å forklare strukturen til Claude hver gang, legger du den inn i en skill.
Slik ser en slik skill ut:

```
---
name: statusrapport-maned
description: Lager månedlig statusrapport i Word-format med
 fast seksjonstruktur. Bruk når du sier «statusrapport»,
 «månedlig rapport» eller «lag rapport for måneden».
metadata:
 author: Ditt navn
 version: 1.0.0
---

# Månedlig statusrapport

Bruk docx-skillen til å lage dokumentet.

## Struktur

Dokumentet skal alltid ha disse seksjonene i denne rekkefølgen:

1. Sammendrag (maks 100 ord)
2. Hva er fullført denne måneden (kulepunktliste)
3. Hva er forsinket og hvorfor (kulepunktliste)
4. Plan for neste måned (nummerert liste)
5. Behov for beslutninger eller ressurser

Spør bruker om måneden og prosjektnavn hvis ikke oppgitt.
Lagre som statusrapport-YYYY-MM.docx.
```

Legg merke til linjen «Bruk docx-skillen til å lage dokumentet». Claude vet da at den skal kombinere instruksjonene dine med Anthropics docx-skill for selve filgenereringen.

> **Tips:** *Skills kan samarbeide. Din skill beskriver innholdet. Anthropics skill håndterer den tekniske filgenereringen. Du trenger ikke bygge det tekniske selv.*

## Tilpasse en ferdig skill

En annen tilnærming er å ta en av Anthropics ferdige skills og utvide den for ditt bruk. Du kopierer mappen, gir den et nytt navn og legger til din egen mal i references/.

```
# Kopier docx-skillen som utgangspunkt
cp -r ~/.claude/skills/docx ~/.claude/skills/vart-dokument
```

Så endrer du beskrivelsen i SKILL.md til å matche ditt spesifikke bruk, og legger til en mal:

```
## Mal

Bruk malen i references/mal.md som utgangspunkt.
Behold overskrifter og seksjonsstruktur fra malen.
Fyll inn innhold basert på brukerens input.
```

**Øvelse 7:**

1. Hent docx-skillen fra github.com/anthropics/skills og legg den i ~/.claude/skills/.
2. Kopier mappen og gi kopien et navn som beskriver dokumenttypen din, for eksempel vart-motereferat.
3. Åpne SKILL.md i den kopierte mappen. Endre name til det nye mappenavnet og skriv en ny description med dine egne triggere.
4. Lag en references/-mappe og legg inn en enkel mal som beskriver hvordan dokumentet ditt alltid skal se ut – seksjoner, overskrifter, fast innhold.
5. Pek til malen fra instruksjonsdelen i SKILL.md. Test i Claude Code: be Claude lage et dokument og sjekk at riktig skill aktiveres.

## Husk dette



- Anthropic tilbyr ferdige skills for docx, pptx, xlsx og pdf — hent fra github.com/anthropics/skills.
- Dokument-skills beskriver innhold og struktur. Den tekniske filgenereringen håndteres av Claudes innebygde evner.
- Skills kan samarbeide: din skill styrer innholdet, Anthropics skill lager filen.
- Tilpass ferdige skills ved å kopiere mappen, endre beskrivelsen og legge til din egen mal i references/.

# Arbeidsflytautomatisering
# Kapittel 8

Den overskriften er en munnfull, men likevel viktig. Dokument-skills lager én ting. Arbeidsflyt-skills gjør flere ting i rekkefølge og tar beslutninger underveis. Det er her skills begynner å likne på en ekte kollega som kan jobbe selvstendig fra start til slutt.

## Hva som skiller en arbeidsflyt-skill

En dokument-skill tar input og produserer én fil. En arbeidsflyt-skill kan stille spørsmål, vente på svar, sjekke at noe er i orden, produsere flere outputs og endre retning basert på hva den finner.

Tenk på forskjellen slik:

| Dokument-skill | Arbeidsflyt-skill |
|---|---|
| Én input, én output | Mange steg, flere outputs |
| Lineær prosess | Beslutninger underveis |
| Claude spør én gang | Claude kan spørre flere ganger |
| Ferdig når filen er lagret | Ferdig når hele prosessen er gjennomført |

## Tre eksempler

**Kodegjennomgang med fast metodikk**

En skill som går gjennom kode i en bestemt rekkefølge: les filen, sjekk for vanlige feil, vurder lesbarhet, foreslå forbedringer. Fast metodikk betyr at gjennomgangen alltid er konsistent og ikke avhengig av hva Claude tilfeldigvis fokuserer på.

```
## Gjennomgang

1. Les filen bruker peker til.
2. Sjekk for disse feilkategoriene i rekkefølge:
   a. Sikkerhetsproblemer (vis liste fra references/sikkerhet.md)
   b. Ytelsesproblemer
   c. Lesbarhet og navngivning
3. For hvert funn: beskriv problemet, vis linjenummer,
   foreslå konkret løsning.
4. Avslutt med en prioritert liste: kritisk / bør fikses / valgfritt.
5. Spør: «Vil du at jeg skal fikse noen av disse nå?»
   Hvis ja: gå til steg 6.
   Hvis nei: avslutt.
6. Gjør endringene bruker velger og vis diff.
```

**Innholdsplanlegging med flere outputs**

En skill som tar et tema og produserer flere ting: en innholdsplan, tittelforslag og et utkast til det første innlegget. Tre outputs fra én forespørsel.

```
## Prosess

1. Spør: «Hva er temaet og hvem er målgruppen?»
2. Lag en innholdsplan med fem innlegg:
  - Tittel
  - Vinkel (hva er poenget?)
  - Nøkkelord
3. Vis planen og spør: «Vil du endre noe
  før jeg skriver det første innlegget?»
4. Skriv utkast til det første innlegget
  basert på godkjent plan.
5. Lagre innholdsplan som innholdsplan.docx
  og utkastet som innlegg-01.docx.
```

**Onboarding-sekvens**

En skill som stiller spørsmål om hva en ny bruker trenger og produserer tilpasset materiale, for eksempel en velkomstmail, en oversikt over verktøy og en liste over første steg. Outputet avhenger av svarene.

```
## Steg

1. Spør: «Hva heter personen og hvilken rolle starter de i?»
2. Spør: «Hvilke verktøy skal de bruke fra dag én?»
3. Basert på rolle og verktøy:
  Hvis teknisk rolle: inkluder oppsettsveiledning
    fra references/teknisk-oppsett.md
  Hvis ikke-teknisk rolle: hopp over teknisk oppsett
4. Lag tre dokumenter:
  - Velkomstmail (klar til å sende)
  - Verktøyoversikt tilpasset rollen
  - Første-uke-plan med daglige mål
```

**Validering underveis**

Det viktigste grepet i en arbeidsflyt-skill er å bygge inn stopp-punkter. I stedet for at Claude ruser gjennom alle steg uten å spørre, legger du inn steder der den venter på bekreftelse.
Dette gjøres med enkle instruksjoner:

```
1. Samle informasjon fra bruker.
2. Lag et utkast basert på informasjonen.
3. Vis utkastet og spør: Er dette riktig før jeg fortsetter?
  Vent på svar.
  Hvis nei: be bruker presisere og gjenta steg 1-2.
  Hvis ja: gå til steg 4.
4. Fullfør og lagre.
```

Stopp-punkter er spesielt viktige i steg der en feil tidlig i prosessen vil ødelegge alt som kommer etter. Bedre å spørre én gang enn å produsere feil output i ti steg.

> **Tips:** *Ikke legg inn stopp-punkt etter hvert eneste steg. Det gjør flyten treig og frustrerende. Bruk dem der en feil faktisk ville vært kostbar.*

## Når en skill er blitt for stor

Arbeidsflyt-skills kan vokse seg store. Det er tre tegn på at det er på tide å dele opp:

- SKILL.md er over én side. Lange instruksjoner er vanskelige å vedlikeholde og øker risikoen for at Claude mister tråden.
- Skillen dekker to distinkt forskjellige oppgaver. Hvis du kan beskrive skillen som «den gjør X og Y», er det to skills.
- Beskrivelsen er blitt bred for å fange alle tilfellene. Da aktiveres den på feil tidspunkt.

Løsningen er å dele. En skill for planlegging, en for skriving, en for gjennomgang. De kan referere til hverandre i instruksjonene:

```
5. Når utkastet er ferdig: si til bruker at de kan
   be om kodegjennomgang for å gå videre.
```

> **Øvelse 8:**
>
> 1. Velg en oppgave du gjør jevnlig som har minst tre steg og ett sted der utfallet avhenger av et svar eller en vurdering.
> 2. Skriv instruksjonene som nummererte steg. Legg inn minst ett stopp-punkt med «Spør bruker om X. Hvis ja: gå til steg N. Hvis nei: gjør Y.»
> 3. Skriv frontmatter med name og description. Husk: under 200 tegn, konkrete triggere.
> 4. Test i Claude Code. Be Claude utføre oppgaven og se om den faktisk stopper opp på stopp-punktet og venter på svar.
> 5. Juster instruksjonene basert på hva du observerer.

## Husk dette 

- Arbeidsflyt-skills har flere steg, beslutninger underveis og kan produsere flere outputs.
- Stopp-punkter med bekreftelse er spesielt viktige der en tidlig feil ødelegger resten.
- En skill som er blitt for stor bør deles opp. Én oppgave per skill.
- Skills kan referere til hverandre i instruksjonene for å lenke arbeidsflyter.

# Skills og MCP
# Kapittel 9

De to foregående kapitlene handlet om skills som jobber alene. Dette kapittelet handler om skills som jobber sammen med noe annet, nemlig MCP.

MCP står for Model Context Protocol og er en måte å koble Claude til eksterne tjenester på. Google Drive, Slack, kalender, databaser eller e-post. Claude får tilgang til verktøyene, men skills bestemmer hvordan den bruker dem.

## Kjøkkenet og oppskriften

I kapittel 2 nevnte vi kort at MCP og skills fyller to forskjellige roller. Nå er det verdt å se nærmere på hva det betyr i praksis.

MCP er kjøkkenet: utstyr, råvarer og tilgang til det du trenger. Google Drive-MCP gir Claude tilgang til filene dine. Slack-MCP gir den tilgang til meldingene dine. Men tilgang er ikke det samme som å vite hva man skal gjøre med det.

Skills er oppskriften: steg for steg, hva som skal gjøres, i hvilken rekkefølge og hvordan resultatet skal se ut. Uten oppskrift kan en kokk improvisere, men resultatet er ikke konsistent. Med oppskrift får du det samme hver gang.

> **Tips:** *Uten skill bruker Claude MCP-verktøyet slik den tror du vil ha det. Med skill bruker den det slik du faktisk vil ha det.*

## Hva MCP-skill gjør

En MCP-skill er en vanlig skill med ett tillegg: instruksjonene refererer til MCP-verktøy ved navn. Claude vet da at den skal kalle på det verktøyet som en del av arbeidsflyten.

Her er et eksempel på en skill som bruker Google Drive:

```
---
name: ukesoppsummering-drive
description: Henter dokumenter fra Google Drive fra siste uke,
 oppsummerer innholdet og lager en statusrapport.
 Bruk når du sier «ukesoppsummering», «hva skjedde denne uken»
 eller «lag ukesstatus fra Drive».
metadata:
 author: Ditt navn
 version: 1.0.0
---
# Ukesoppsummering fra Google Drive
## Forutsetning

Google Drive MCP må være koblet til og autentisert.

## Steg
```

```
1. Bruk Google Drive MCP til å søke etter dokumenter
   endret de siste 7 dagene.
2. List opp dokumentene med tittel og dato.
3. Spør: «Vil du at jeg skal lese gjennom alle,
   eller bare noen av dem?»
4. Les innholdet i de valgte dokumentene.
5. Skriv en oppsummering per dokument (maks 3 setninger).
6. Lag en samlet statusrapport med alle oppsummeringene.
7. Lagre rapporten som ukesoppsummering-YYYY-MM-DD.docx.
```

Legg merke til «Bruk Google Drive MCP» i steg 1. Det er alt som skal til for at Claude forstår at den skal kalle på MCP-verktøyet. Resten er vanlige skill-instruksjoner.

## Hva du trenger for å komme i gang

Skills-siden av en MCP-skill er enkel, det er det du nå kan. Det som krever litt mer arbeid er MCP-oppsettet. For å bruke MCP i Claude Code trenger du:

- En MCP-server installert og konfigurert. Anthropic vedlikeholder en liste over offisielle MCP-servere på github.com/anthropics/mcp.
- Autentisering satt opp. De fleste MCP-servere krever at du logger inn eller oppgir en API-nøkkel første gang.
- MCP aktivert i Claude Code. Dette gjøres i konfigurasjonsfilen .claude/config.json i prosjektmappen din.

> **Tips:** *MCP-konfigurasjon er utenfor denne bokens scope, men Anthropics offisielle dokumentasjon på docs.anthropic.com/mcp dekker det grundig. Skill-siden er den samme uansett hvilken MCP-server du bruker.*

## Feilhåndtering i MCP-skill

MCP-tilkoblinger kan feile. Nettverket er nede, autentiseringen er utløpt, tjenesten er utilgjengelig. En god MCP-skill håndterer dette eksplisitt:

```
1. Bruk Google Drive MCP til å hente dokumenter.
   Hvis MCP ikke svarer eller returnerer feil:
   - Fortell bruker: «Jeg får ikke kontakt med Google Drive.»
   - Be dem sjekke at MCP er koblet til.
   - Avslutt.
2. Fortsett bare hvis steg 1 lyktes.
```

Uten slik håndtering vil Claude prøve å fortsette selv om MCP feiler og produsere meningsløst output.

## Når MCP-skill ikke er nødvendig

Ikke alle MCP-brukstilfeller trenger en skill. Hvis du bare vil gjøre noe én gang, er det raskere å be Claude direkte:

> Bruk Google Drive til å finne filen «budsjett Q3»
> og oppsummer innholdet.

En skill lønner seg når du gjør det samme jevnlig og vil ha konsistent metodikk hver gang. Er det en engangsoppgave, bruk chatten.

**Øvelse 9:**

1. Velg en ekstern tjeneste du bruker jevnlig – e-post, kalender, Google Drive, Slack eller annet.
2. Skriv instruksjonsdelen til en skill som bruker denne tjenesten. Du trenger ikke ha MCP satt opp ennå – fokuser på skill-logikken.
3. Inkluder minst ett steg der Claude kaller på MCP-verktøyet ved navn, og ett steg der den håndterer en mulig feil.
4. Skriv frontmatter med name og description. Husk triggere som er spesifikke for akkurat denne tjenesten og oppgaven.
5. Les gjennom: er instruksjonene tydelige nok til at Claude vet nøyaktig hva den skal gjøre selv om MCP svarer med uventet innhold?

## Husk dette



- MCP gir tilgang. Skills bestemmer hvordan tilgangen brukes.
- En MCP-skill er en vanlig skill der instruksjonene refererer til MCP-verktøy ved navn.
- Feilhåndtering er viktig: skill bør eksplisitt si hva Claude skal gjøre hvis MCP feiler.
- Bruk skill for jevnlige oppgaver med fast metodikk. Bruk chatten for engangsoppgaver.

# Installere og teste
# Kapittel 10

Du har lært å bygge skills. Nå skal vi se på hva som skjer etterpå, hvordan du installerer dem riktig, bekrefter at de er tilgjengelige og tester at de faktisk gjør det de skal. Testing er det steget de fleste hopper over. Det er også grunnen til at mange skills aldri fungerer optimalt.

**Tre steder å installere**

Hvor du lagrer en skill bestemmer hvem som har tilgang til den og i hvilke sammenhenger den er tilgjengelig:

| Plassering | Tilgjengelig for | Bruk når |
|---|---|---|
| ~/.claude/skills/ | Alle prosjekter på maskinen din | Skills du bruker overalt |
| .claude/skills/ | Bare det aktuelle prosjektet | Skills knyttet til ett prosjekt |
| Admin-panel (organisasjon) | Alle brukere i organisasjonen | Felles skills for et team |

De fleste skills hører hjemme i ~/.claude/skills/. Bruk prosjektmappen bare hvis skillen er tett knyttet til ett spesifikt prosjekt og ikke gir mening utenfor det.

## Bekreft at skillen er lastet

Etter installasjon: start Claude Code og kjør:

```
/context
```

Du skal se skillnavnet ditt i listen. Hvis det ikke er der, er det som regel én av tre årsaker:

- Feil mappenavn. Sjekk at mappen bruker kebab-case og ikke inneholder mellomrom eller store bokstaver.
- Feil filnavn. Filen må hete nøyaktig SKILL.md med store bokstaver.
- Kontekstbudsjettet er fullt. /context viser en advarsel hvis skills er ekskludert. Forkort beskrivelsene i andre skills.

## Teste systematisk

En skill som fungerer én gang er ikke nødvendigvis en skill som fungerer konsistent. Test den med minst fem forskjellige formuleringer, både de du forventer skal aktivere den og noen du forventer ikke skal gjøre det.

**Test 1: Aktiveres på riktige triggere**

Prøv alle formuleringene du har i description-feltet, pluss naturlige varianter:

```
# Eksempel for motereferat-skillen
lag referat
møtenotat fra i dag
skriv opp møtet
vi hadde møte, kan du lage referat?
trenger referat fra teammøtet
```

Sjekk i tankekjeden at skillen faktisk lastes inn. Du ser det som en linje der Claude leser SKILL.md.

**Test 2: Aktiveres ikke på feil triggere**

Prøv formuleringer som ligner, men som ikke bør aktivere skillen:

```
# Disse skal ikke aktivere motereferat-skillen
lag en rapport
skriv et sammendrag
hva skjedde på møtet?
```

Hvis skillen aktiveres på disse, er description-feltet for bredt. Gjør triggerne mer spesifikke.

**Test 3: Følger instruksjonene**

Gi Claude en realistisk oppgave og sjekk at outputet har riktig struktur — alle seksjoner er med, rekkefølgen stemmer, format er som spesifisert.

**Test 4: Håndterer manglende input**

Hva skjer hvis du gir minimal informasjon?

```
lag referat
```

Spør Claude om det som mangler, eller gjetter den og fyller inn selv? Instruksjonene bør si eksplisitt hva Claude skal gjøre i dette tilfellet.

**Test 5: Konsistent over tid**

Kjør samme forespørsel tre ganger og sammenlign outputet. Strukturen skal være lik selv om innholdet varierer. Hvis outputet er veldig forskjellig, er instruksjonene for vage.

## Iterere basert på observasjon

Når noe ikke fungerer, er det enten beskrivelsen eller instruksjonene som er problemet , sjelden begge.

- Skillen aktiveres ikke eller aktiveres feil: problemet er i description-feltet. Juster triggerne.
- Skillen aktiveres riktig men produserer feil output: problemet er i instruksjonsdelen. Gjør stegene mer konkrete.
- Skillen aktiveres på alt: description er for bred. Gjør oppgavebeskrivelsen mer spesifikk.
- Claude mister tråden midt i arbeidsflyten: instruksjonene er for lange. Del opp eller flytt detaljer til references/.

**Tips:** *Endre én ting av gangen. Hvis du endrer både description og instruksjonene samtidig, vet du ikke hva som løste problemet.*

**Øvelse 10:**

1. Åpne skillen du lagde i øvelse 3 eller 7 i Claude Code.
2. Kjør /context og bekreft at skillnavnet er synlig i listen.
3. Test med fem forskjellige formuleringer som bør aktivere skillen. Noter hvilke som fungerer og hvilke som ikke gjør det.
4. Test med to formuleringer som ikke bør aktivere skillen. Sjekk at den ikke aktiveres.
5. Gi Claude en realistisk oppgave og sammenlign outputet med instruksjonene dine. Er alle steg fulgt? Juster det som mangler.

## Husk dette



- Personlige skills lagres i ~/.claude/skills/, prosjektskills i .claude/skills/ i prosjektmappen.
- /context bekrefter at skillen er lastet. Ikke der? Sjekk mappenavn, filnavn og kontekstbudsjettet.
- Test med riktige og feil triggere, manglende input og gjentatte kjøringer.
- Aktiveres ikke: juster description. Feil output: juster instruksjonene. Endre én ting av gangen.

# Skills i Claude.ai og Claude Code
## Kapittel 11

En av de praktiske fordelene med skills som åpen standard er at du lager én SKILL.md og bruker den flere steder. Fungerer overalt betyr ikke fungerer identisk overalt. Det er noen forskjeller du bør kjenne til.

**Hva som er likt**

Formatet er det samme. SKILL.md med YAML-frontmatter, instruksjoner i Markdown, støttefiler i undermapper. En skill du har bygget i kapittel 4 og 5 kan brukes direkte i claude.ai uten endringer, forutsatt at den ikke bruker scripts. Scripts og filsystemtilgang blir stoppet i nettleser av sikkerhetsmessige årsaker.

Beskrivelsen fungerer likt. Claude scanner tilgjengelige skills og aktiverer dem basert på relevans, uavhengig av om du er i terminalen eller nettleseren.

**Hva som er forskjellig**

Forskjellen handler om miljø. Claude Code kjører på maskinen din med tilgang til filsystemet. claude.ai kjører i nettleseren uten det.

| Funksjon | Claude Code | Claude.ai |
|---|---|---|
| SKILL.md instruksjoner | ✓ | ✓ |
| references/-filer | ✓ | ✓ |
| assets/-filer | ✓ | ✓ |
| scripts/ (Python, Bash) | ✓ | ✗ |
| Filsystemtilgang | ✓ | ✗ |
| Terminalkommandoer | ✓ | ✗ |
| Dokument-skills (docx, pptx) | ✓ | ✓ |
| MCP-integrasjoner | ✓ | Avhenger av MCP |

> **Tips:** *Har du bygget skills uten scripts/ og uten filsystemtilgang, fungerer de allerede i claude.ai. De fleste skills i del 2 og 3 er av denne typen.*

## Installere i claude.ai

I claude.ai laster du opp skills via innstillingene. Slik gjør du det:

1. Gå til Settings > Capabilities > Skills.
2. Pakk skill-mappen din som en zip-fil. Mappen med SKILL.md skal ligge øverst i zip-filen.
3. Last opp zip-filen.
4. Start en ny samtale og bekreft at skillen er aktiv.

Slik lager du zip-filen på Mac og Windows:

```
# Mac/Linux — pakk mappen i terminalen
cd ~/.claude/skills
zip -r motereferat.zip motereferat/

# Windows — høyreklikk på mappen i Utforsker
# og velg Komprimer til ZIP-fil
```

## Samme skill eller to separate

Noen ganger fungerer én skill godt i begge kontekster. Andre ganger gir det mer mening å lage to separate skills, én optimert for Claude Code og én for claude.ai.

Bruk én skill når:

- Skillen ikke bruker scripts/ eller filsystemtilgang.
- Arbeidsflyten er den samme uavhengig av plattform.
- Du vil vedlikeholde én fil, ikke to.

Lag to separate skills når:

- Claude Code-versjonen bruker scripts og trenger en enklere versjon for nettleseren.
- Arbeidsflyten er fundamentalt forskjellig på de to plattformene.
- Du vil ha ulike triggere for ulike kontekster.

Navngi dem da slik at det er tydelig:

```
motereferat/        ← Fungerer begge steder
motereferat-cli/    ← Bare Claude Code
motereferat-web/    ← Bare claude.ai
```

## Live oppdatering i Claude Code

En praktisk detalj: skills i .claude/skills/-mappen i et prosjekt oppdateres automatisk mens du jobber. Du trenger ikke starte Claude Code på nytt etter at du har endret en SKILL.md. Personlige skills i ~/.claude/skills/ krever en omstart.

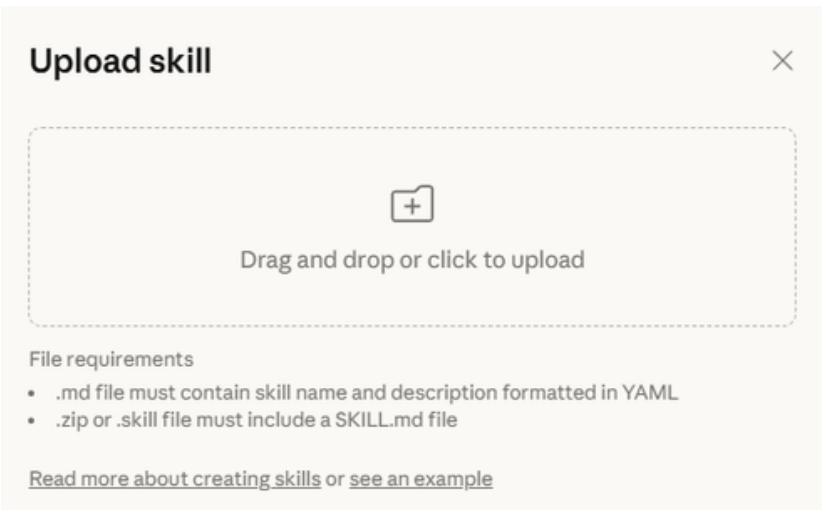
**Øvelse 11:**

1. Velg en skill du har laget som ikke bruker scripts/ – for eksempel den fra øvelse 3.
2. Pakk skill-mappen som zip-fil.
3. Gå til claude.ai > Settings > Capabilities > Skills og last opp zip-filen.
4. Start en ny samtale og test med en formulering som bør aktivere skillen.
5. Sammenlign oppførselen med Claude Code. Er outputet konsistent?

## Husk dette



- Samme SKILL.md-format fungerer i Claude Code og claude.ai. Forskjellen er miljøet, ikke formatet.
- Scripts og filsystemtilgang fungerer bare i Claude Code. Rene instruksjonsskills fungerer begge steder.
- I claude.ai laster du opp skills som zip-fil via Settings > Capabilities > Skills.
- Én skill er enklest å vedlikeholde. To separate gir mer fleksibilitet når behovene er forskjellige.




Bilde: Last opp skillen som .zip eller .md i claudei.ai

# Organisasjonsskills
# Kapittel 12

Hittil har vi bygget skills for deg selv. Dette kapittelet handler om neste steg: skills som deles med et helt team eller en hel organisasjon.

Forskjellen er ikke teknisk. Formatet er det samme. Forskjellen er at du nå bygger for folk som ikke kjenner skillen din, som formulerer seg annerledes enn deg, og som ikke vil lese dokumentasjonen.

## Distribusjon via admin-panelet

Med Team- eller Enterprise-plan kan en administrator distribuere skills til alle brukere i organisasjonen via admin-panelet i claude.ai. Brukerne trenger ikke installere noe selv. Skills dukker opp automatisk neste gang de starter Claude.

For deg som bygger skills betyr det at én person vedlikeholder og alle nyter godt av oppdateringer. Endrer du SKILL.md og laster opp på nytt, får alle den nye versjonen.

> **Tips:** *Personlige skills krever ingen plan. Organisasjonsskills via admin-panelet krever Team- eller Enterprise-plan.*

## Bygge for andre er annerledes

Når du bygger en skill for deg selv, vet du nøyaktig hva du skriver til Claude. Triggerne dine passer perfekt fordi du lagde dem.

Når du bygger for kolleger, vet du ikke hva de skriver. Noen skriver kortfattet, noen skriver lange setninger. Noen bruker fagterminologi, noen bruker hverdagsspråk. En skill som bare aktiveres på nøyaktig de frasene du har i description-feltet vil svikte mange brukere.

Tre grep som hjelper:

**1. Bredere triggere**

Legg inn flere varianter i description-feltet. Test med kollegene dine og spør: hva ville du skrevet for å be om dette?

```
# Smal (bare deg)
description: Lager møtereferater. Bruk ved: lag referat, møtenotat.

# Bred (for team)
description: Lager strukturerte møtereferater i Word-format.
 Bruk når bruker sier lag referat, møtenotat, skriv opp
 møtet, referat fra i dag, kan du ta referat, hva ble
 bestemt på møtet, oppsummer diskusjonen.
```

**2. Mer robust feilhåndtering**

En personlig skill kan anta at du alltid gir riktig input. En organisasjonsskill bør håndtere manglende informasjon eksplisitt og stille oppklarende spørsmål i stedet for å gjette.

```
1. Sjekk at bruker har oppgitt dato, deltakere og tema.
   Mangler noe: spor om det spesifikke feltet.
   Ikke fortsett før alle felt er fylt inn.
```

**3. Enklere instruksjoner**

Du kan tillate deg komplekse instruksjoner når du er eneste bruker. For et team bør skillen være enklere og mer forutsigbar. Fjern kanttilfeller du aldri har sett, og behold kjernen.

## Versjonering

Metadata-feltet med version er spesielt nyttig når skills deles. Når du oppdaterer en skill, hever du versjonsnummeret. Det gir deg og teamet oversikt over hva som er endret og når.

En enkel konvensjon:

```
metadata:
  author: Ditt navn
  version: 1.0.0  # Første versjon
  version: 1.1.0  # Ny funksjon lagt til
  version: 1.1.1  # Liten fiks
  version: 2.0.0  # Stor endring, instruksjoner omskrevet
```

Legg gjerne til en endringslogg i SKILL.md under frontmatter:

```
## Endringslogg

v1.1.0: La til steg for oppfølgingspunkter med frist
v1.0.0: Første versjon
```

## Test med ekte brukere

Den viktigste forskjellen mellom en personlig skill og en organisasjonsskill er testingen. Du kan ikke teste en organisasjonsskill bare med deg selv.

Be to til tre kolleger prøve den før du distribuerer. Gi dem ingen instruksjoner om hva de skal skrive. Se hva som aktiverer skillen og hva som ikke gjør det. Juster triggerne basert på hva du ser.

**Tips:** *Den beste testen er å se noen andre bruke skillen for første gang uten at du forklarer noe. Det du ser da, er hva som faktisk mangler.*

## Når organisasjonsskills ikke er riktig

Ikke alt egner seg som organisasjonsskill. Vurder to ganger først:

- Svært personlige skills. En skill for din personlige skrivestil passer ikke for kolleger med annen stil.
- Skills under utvikling. Distribuer ikke noe du fortsatt eksperimenterer med. Folk stoler på det som ligger der.
- For spesifikke skills. En skill som bare gjelder ett prosjekt eller en rolle trenger ikke distribueres til hele organisasjonen.

**Øvelse 12:**

1. Åpne en skill du har laget og som du tror kolleger ville hatt nytte av.
2. Les description-feltet. Skriv ned tre alternative måter en kollega kunne formulert den samme forespørselen på. Legg dem til som triggere.
3. Les instruksjonene. Finn minst ett sted der du antar noe om inputet. Legg til et steg som spør om det mangler.
4. Legg til metadata med author og version: 1.0.0.
5. Del SKILL.md med en kollega og be dem prøve den uten forklaring. Se hva som skjer og noter hva som må justeres.

## Husk dette



- Organisasjonsskills distribueres via admin-panelet og krever Team- eller Enterprise-plan.
- Bygg for ukjente: bredere triggere, robust feilhåndtering, enklere instruksjoner.
- Bruk versjonsnummer i metadata og hold en endringslogg i SKILL.md.
- Test alltid med ekte brukere for distribusjon. Det du ser da er hva som faktisk mangler.

# Teste at skillen fungerer
## Kapittel 13

Du har skrevet SKILL.md. Du har lagt inn instruksjoner, triggere og eventuelt støttefiler. Nå kommer det spørsmålet mange hopper over: virker den egentlig?

Testing av skills handler om to ting: om skillen aktiveres når den skal, og om den gjør det den lover når den først er aktiv. Begge deler kan måles, både med tall og med observasjon.

## Kvantitative mål

Kvantitative mål gir deg konkrete tall å sammenligne over tid. De tre mest nyttige er triggerrate, verktøykall og feilrate.

**Triggerrate: 90 prosent som mål**

Triggerraten måler hvor ofte skillen aktiveres når den burde. Du tester dette ved å sende ti ulike formuleringer av den samme typen forespørsel og telle hvor mange ganger skillen faktisk aktiveres.

90 prosent er et godt mål. Det betyr at ni av ti naturlige formuleringer utløser skillen. Kommer du under 70 prosent, er description-feltet for smalt. Kommer du over 95 prosent, bør du sjekke om skillen aktiveres i situasjoner der den ikke burde.

> **Tips:** *Send de samme ti setningene før og etter at du justerer description-feltet. Slik ser du om endringen faktisk hjelper.*

**Færre verktøykall**

En godt skrevet skill gjør at Claude trenger færre verktøykall for å fullføre oppgaven. Instruksjonene er klare nok til at Claude ikke må spørre, lete eller prøve på nytt. Gjør den samme oppgaven to ganger: én gang uten skill, én gang med. Tell antall verktøykall i Claude Code sin tankekjede. Hvis skillen ikke reduserer antallet, er instruksjonene uklare eller for generelle.

**Null feilede API-kall**

Feilede API-kall betyr at skillen ber Claude gjøre noe som ikke virker — kalle et verktøy som ikke finnes, bruke feil parametre eller referere til filer som mangler. Målet er null slike feil.

Ser du feil i tankekjeden, les instruksjonene på nytt og finn stedet der Claude misforstår. Er verktøynavnet skrevet riktig? Er stien til støttefilene korrekt? Er rekkefølgen på stegene logisk?

## Kvalitative mål

Tall forteller deg om skillen virker. Observasjon forteller deg om den er god. De to hører sammen.

**Brukere trenger ikke omdirigere**

Et tydelig tegn på at en skill ikke fungerer: brukeren må korrigere Claude underveis. “Nei, ikke slik, gjør det på den måten i stedet.” Hvis det skjer jevnlig, er instruksjonene uklare på det punktet.

Når du ser at du omdirigerer Claude på samme punkt to ganger, skriv det ned. Det er en instruksjon som mangler i SKILL.md.

**Konsistente resultater**

En god skill gir omtrent det samme resultatet hver gang, uavhengig av hvordan forespørselen er formulert. Hvis outputet varierer mye fra gang til gang, er instruksjonene for løse.

Test dette ved å kjøre den samme oppgaven tre ganger i separate samtaler. Sammenlign outputet. Er strukturen den samme? Er tonen lik? Bruker Claude de riktige malene? Variasjon på innhold er greit, variasjon på form er et tegn på at instruksjonene må spesifiseres.

> **Tips:** *Variasjon på form er alltid et signal, aldri bare tilfeldighet. Finn steget i instruksjonene som ikke er presist nok, og skriv det om.*

## Iterere: når og hvordan

En skill er ikke ferdig første gang du lagrer den. Den blir bedre gjennom bruk. Det er ikke en svakhet ved skills, det er slik alle gode arbeidsmetoder utvikler seg.

Hold en enkel logg over hva du endrer og hvorfor. Ikke for andres skyld, for din egen. Når du en dag lurer på hvorfor en instruksjon er skrevet på en bestemt måte, vil du være glad for at du noterte det.

En tommelfingerregel: juster når du merker at du omdirigerer Claude på samme punkt to ganger. Det er signalet om at instruksjonen må skrives om, ikke at du må forklare bedre i samtalen.

> **Øvelse 13:**
>
> 1. Åpne en skill du har laget og som du tror kolleger ville hatt nytte av.
> 2. Les description-feltet. Skriv ned tre alternative måter en kollega kunne formulert den samme forespørselen på. Legg dem til som triggere.
> 3. Les instruksjonene. Finn minst ett sted der du antar noe om inputet. Legg til et steg som spør om det mangler.
> 4. Legg til metadata med author og version: 1.0.0.
> 5. Del SKILL.md med en kollega og be dem prøve den uten forklaring. Se hva som skjer og noter hva som må justeres.

## Husk dette



1. Velg en skill du har laget. Skriv opp ti ulike måter du eller en kollega ville formulert den samme forespørselen på.
2. Send alle ti til Claude Code. Sjekk tankekjeden og tell hvor mange ganger skillen aktiveres.
3. Kom du under 9 av 10? Les description-feltet og legg til formuleringene som ikke trigget. Test på nytt.
4. Kjør den samme oppgaven tre ganger i separate samtaler. Sammenlign outputet. Noter hva som varierer på form og skriv det inn i instruksjonene.

# Distribuere og dele
## Kapittel 14

En skill som bare lever på din maskin, er nyttig. En skill som deles med andre er verdifull. Dette kapittelet handler om hvordan du gjør begge deler og hva som finnes av fellesskap og infrastruktur rundt skills.

## Github

Den enkleste måten å dele en skill på er via GitHub. Du oppretter et offentlig repositorium, legger inn skill-mappen med SKILL.md og eventuelle støttefiler, og deler lenken. Andre kan laste ned mappen og legge den i sin egen skills-katalog.

GitHub gir deg versjonskontroll gratis. Hver endring i SKILL.md logges med tidspunkt og kommentar. Du kan rulle tilbake til en tidligere versjon hvis noe går galt, og andre kan se nøyaktig hva som er endret mellom versjonene.

En god README.md i repoet bør forklare hva skillen gjør, hvilke triggere den responderer på og eventuelle avhengigheter. Jo lettere det er for andre å forstå skillen uten å åpne SKILL.md, jo flere vil bruke den.

> **Tips:** *En avhengighet (dependency) er et eksternt program eller en pakke som skillen din ikke inneholder selv, men som må være installert på maskinen for at skillen skal fungere. Skillen antar at verktøyet finnes og hvis det ikke gjør det, feiler den.*

## agentskills.io

agentskills.io er et dedikert register for skills. Her kan du publisere skillen din med beskrivelse, kategori og versjonsnummer, og andre kan søke opp skills etter bruksområde.

Tenk på det som en pakkeforvalter for skills, samme prinsipp som npm for JavaScript eller pip for Python.

Fordelen med agentskills.io fremfor GitHub er oppdagbarhet. En skill du publiserer der kan finnes av folk som aldri ville søkt på GitHub. Du når et bredere publikum med mindre innsats.

## Claude Code Plugin Marketplace

Anthropic har bygget en markedsplass direkte inn i Claude Code. Du får tilgang til den via plugin-kommandoen i terminalen. Anthropics egen offisielle markedsplass er allerede tilgjengelig uten ekstra oppsett. Når du åpner plugin-kommandoen, får du et fanebasert grensesnitt der du kan bla gjennom tilgjengelige plugins, installere dem og se eventuelle feil. Du kan også legge til tredjeparts markedsplasser. Skillen lastes ned og plasseres automatisk i riktig mappe. Installasjon skjer med to kommandoer:

```
/plugin marketplace add <repo>
/plugin install <plugin-navn>
```

Skillen lastes ned og plasseres automatisk i riktig mappe. Du trenger ikke å finne frem til katalogen selv.

## Open source og Apache 2.0

Skills-standarden er publisert under Apache 2.0-lisensen. Det betyr at du fritt kan bruke, endre og redistribuere skills, også i kommersielle sammenhenger, så lenge du beholder lisensteksten. Du trenger ikke spørre om lov.

For deg som jobber i skolen betyr dette at en skill du har laget til norskfaget kan deles fritt med alle norsklærere i landet uten juridiske komplikasjoner. Ingen trenger å søke tillatelse eller betale lisensavgift.

Hvis du vil at andre skal kunne bidra til skillen din, legg til en CONTRIBUTING.md i repositoriet med retningslinjer for hva du tar imot av endringer og hvordan pull requests behandles.

## Open source og Apache 2.0

Skills er en ung standard, og fellesskapet rundt dem vokser fort. Det finnes allerede aktive miljøer på Discord, Reddit og GitHub der folk deler skills, diskuterer design og hjelper hverandre med feilsøking.

Det mest verdifulle du kan bidra med er ikke nødvendigvis den mest avanserte skillen. Det er skills som løser vanlige problemer på en enkel og pålitelig måte. En skill for å lage ukentlige møtereferater er langt mer nyttig enn en avansert kodeskilll ingen forstod seg på.

Når du bruker en skill noen andre har laget og den fungerer bra, si ifra. En kommentar, en stjerne på GitHub eller en melding i forumet koster ingenting og betyr mye for den som brukte tid på å lage den.

**Øvelse 14:**

1. Velg en skill du har laget og er fornøyd med. Den bør være testet og ha et versjonsnummer.
2. Opprett et offentlig GitHub-repositorium med navn som samsvarer med skill-mappen.
3. Skriv en README.md som forklarer hva skillen gjør, hvilke triggere den bruker, og hvordan man installerer den.
4. Del lenken med minst én kollega og be om tilbakemelding.

## Husk dette



1. GitHub er enklest for deling og gir versjonskontroll gratis.
2. agentskills.io gir bedre oppdagbarhet for folk utenfor GitHub-miljøet.
3. Plugin Marketplace er innebygd i Claude Code og gir enkel installasjon med én kommando.
4. Apache 2.0 betyr fri bruk, endring og redistribusjon, også kommersielt.
5. Enkle, pålitelige skills for hverdagsbehov er mer verdifulle for fellesskapet enn avanserte skills ingen forstår.

## Appendiks A: Mal for SKILL.md

```yaml
---
name: [Sett inn navnet på din skill her]
description: >
  [En kort beskrivelse av hva skillen gjør.]
  [Eksempler på hva brukeren kan si for å utløse skillen.]
  [Eksempel: "Lag rapport, ukesoppsummering, skriv status..."]
metadata:
  author: [Ditt navn]
  version: 1.0.0
  license: Apache-2.0
---

# Tittel på skillen

## Når skal denne skillen brukes?
Beskriv situasjonen der skillen er relevant. Vær konkret: hvilken type oppgave, hvilken kontekst, hvilke brukere.

## Forutsetninger
- Brukeren har oppgitt [X]
- [Avhengighet] er installert (se README.md)

## Steg for steg
1. Sjekk at nødvendig informasjon er tilstede. Mangler noe, spør om det spesifikke feltet før du fortsetter.
2. [Andre steg]
3. [Tredje steg]
4. Bekreft resultatet med brukeren hvis outputet er langt eller komplekst.

## Output-format
Beskriv hva skillen skal produsere: filtype, struktur, tone, lengde, språk.

## Feilhåndtering
- Mangler dato: spør brukeren om dato før du fortsetter.
- Mangler deltakerliste: bruk «ikke oppgitt» som standardverdi.
- Ukjent filformat: informer brukeren og foreslå alternativer.

## Eksempel
Input: «[Eksempel på brukerinput]»
Output: [Kort beskrivelse av forventet resultat]

## Endringslogg
- v1.0.0: Første versjon
```

**Tips:** *Bruk kun mellomrom (aldri Tab) for innrykk, sett > etter description for tekst over flere linjer, og sørg for at hele YAML-blokken rammes inn av tre bindestreker*

## Appendiks B: Sjekkliste for ny skill

**Før du skriver instruksjonene:**

- [ ] Oppgaven har en fast metodikk som gjentar seg
- [ ] Du kan beskrive den i minst tre konkrete steg
- [ ] Det er tydelig når skillen skal brukes og ikke brukes

**Når du skriver SKILL.md:**

- [ ] Navn er kebab-case og beskrivende (ikke «min-skill» eller «test»)
- [ ] Description-feltet inneholder minst fem triggerfraser i ulike formuleringer
- [ ] Instruksjonene er skrevet for noen som ikke kjenner konteksten din
- [ ] Feilhåndtering dekker de vanligste tilfellene av manglende input
- [ ] Output-format er spesifisert: filtype, struktur, språk, tone
- [ ] Versjonsnummer er satt i metadata (start på 1.0.0)

**Før du distribuerer:**

- [ ] Testet med minst tre ulike formuleringer av samme forespørsel
- [ ] Testet av minst én annen person uten forklaring fra deg
- [ ] Avhengigheter er dokumentert i README.md med installasjonskommando
- [ ] README.md forklarer hva skillen gjør og hvilke triggere den bruker
- [ ] Endringslogg er oppdatert

**Etter distribusjon:**

- [ ] Innhentet tilbakemelding fra brukere etter første uke
- [ ] Justert triggere basert på hva brukere faktisk skriver
- [ ] Versjonsnummer hevet ved endringer

yaml

```
---
name: skill-navn
description: Kort beskrivelse av hva skillen gjør.
 Bruk når bruker sier: lag rapport, ukesoppsummering,
 skriv status, hva skjedde denne uken, kan du lage rapport.
metadata:
 author: Ditt navn
 version: 1.0.0
 license: Apache-2.0
---
# Tittel på skillen
## Når skal denne skillen brukes?
Beskriv situasjonen der skillen er relevant. Vær konkret:
hvilken type oppgave, hvilken kontekst, hvilke brukere.

## Forutsetninger
- Brukeren har oppgitt [X]
- [Avhengighet] er installert (se README.md)

## Steg for steg
1. Sjekk at nødvendig informasjon er tilstede. Mangler noe,
 spør om det spesifikke feltet før du fortsetter.
2. [Andre steg]
3. [Tredje steg]
4. Bekreft resultatet med brukeren hvis outputet er langt
 eller komplekst.

## Output-format
Beskriv hva skillen skal produsere: filtype, struktur,
tone, lengde, språk.

## Feilhåndtering
- Mangler dato: spør brukeren om dato før du fortsetter.
- Mangler deltakerliste: bruk «ikke oppgitt» som standardverdi.
- Ukjent filformat: informer brukeren og foreslå alternativer.

## Eksempel
Input: «Lag referat fra møtet i dag»
Output: Strukturert møtereferat med dato, deltakere,
beslutninger og oppfølgingspunkter.

## Endringslogg
- v1.0.0: Første versjon
```

# Appendiks C: Ordliste

**Avhengighet** (dependency) — Eksternt program eller pakke som må være installert på maskinen for at skillen skal fungere. Eksempler: Node.js, pandoc, LibreOffice. Skal alltid dokumenteres i README.md med versjon og installasjonskommando.

**Description-felt** — Feltet i frontmatter som forteller Claude når skillen er relevant. Inneholder en kort forklaring og triggerfraser. Det viktigste du skriver i hele skillen: en dårlig description betyr at skillen aldri aktiveres, uansett hvor gode instruksjonene er.

**Endringslogg** (changelog) — En logg i SKILL.md som viser hva som er endret mellom versjoner. Gjør det enkelt for andre å forstå hva som er nytt uten å lese hele filen på nytt.

**Frontmatter** — Den strukturerte blokken øverst i SKILL.md, avgrenset av --- over og under. Inneholder navn, beskrivelse og metadata. Leses av Claude ved oppstart og er alltid tilstede i systemet.

**Kebab-case** — Navnekonvensjon der ord skilles med bindestrek og alt er med små bokstaver. Eksempel: ukentlig-rapport, motereferat-norsk. Brukes for skill-mappenavn og name-feltet i frontmatter.

**Modellstyrt** — Aktiveres av Claude selv basert på hva brukeren holder på med, ikke av brukeren via en eksplisitt kommando. Motsetningen er brukerstyrt, som slash-kommandoer.

**Progressiv avsløring** — Prinsippet om at skills lastes inn i tre lag: først beskrivelsen (alltid i kontekst), så instruksjonene (når skillen er relevant), så støttefiler (bare når nødvendig). Gjør det mulig å ha mange skills uten at Claude blir treg.

**Pull request** — En forespørsel om å flette inn endringer i et GitHub-repositorium. Brukes når andre vil bidra til skillen din. Du godkjenner eller avviser endringene før de tas inn.

**Repositorium** (repository) — En mappe på GitHub som inneholder alle filene til en skill, med full endringshistorikk. Forkortes gjerne til «repo».

**Skill** — En mappe med instruksjoner Claude aktiverer automatisk når de er relevante for oppgaven. Består av SKILL.md og valgfrie støttemapper som scripts/, references/ og assets/.

**SKILL.md** — Den eneste obligatoriske filen i en skill-mappe. Inneholder frontmatter øverst og instruksjoner under. Alt annet er valgfritt.

**Triggerfrase** — Et ord eller en setning i description-feltet som signaliserer til Claude at skillen er relevant. Jo flere varierte triggerfraser, jo mer robust er skillen for ulike brukere og formuleringer.

**Versjonskontroll** — System for å spore og lagre alle endringer i en fil over tid. Med Git og GitHub får du full historikk, mulighet til å rulle tilbake, og oversikt over hva som er endret og når.

**Versjonsnummer** (semantic versioning) — Tredelt nummersystem på formen X.Y.Z. X heves ved store endringer som bryter bakoverkompatibilitet, Y ved nye funksjoner, Z ved små feilrettinger. Eksempel: 1.0.0 → 1.1.0 → 1.1.1 → 2.0.0.

# Kilder

## Offisiell dokumentasjon


- Anthropic. Claude Code Docs. https://code.claude.com/docs
- Anthropic. Skills — offisielt GitHub-repositorium. https://github.com/anthropics/skills
- Anthropic. The Complete Guide to Building Skills for Claude (PDF, 29. januar 2026). https://resources.anthropic.com/hubfs/The-Complete-Guide-to-Building-Skill-for-Claude.pdf?hsLang=en
- Anthropic. A Complete Guide to Building Skills for Claude (bloggversjon). https://claude.com/blog/complete-guide-to-building-skills-for-claude
- Anthropic. Official Claude Code Plugins. https://github.com/anthropics/claude-plugins-official
- Anthropic. Claude Cookbook — praktiske guider og kodeeksempler. https://platform.claude.com/cookbook/


## Artikler og guider


- AI Blew My Mind. Claude Skills — 36 eksempler. https://aiblewmymind.substack.com/p/claude-skills-36-examples
- AI Blew My Mind. Claude Cowork Plugins Guide. https://aiblewmymind.substack.com/p/claude-cowork-plugins-guide
- Product Talk. How to Use Claude Code Features. https://www.producttalk.org/how-to-use-claude-code-features/
- DataCamp. How to Build Claude Code Plugins: A Step-by-Step Guide. https://www.datacamp.com/tutorial/how-to-build-claude-code-plugins
- Composio. Improving Your Coding Workflow with Claude Code Plugins. https://composio.dev/blog/claude-code-plugin


## Ressurser og samlinger


- **A**I Templates. Skills og plugin-komponenter. https://www.aitmpl.com/component/skill/docx
- Claude Marketplaces. Plugin-oversikt. https://claudemarketplaces.com
- Script by AI. The Ultimate Claude Code Resource List (2026). https://www.scriptbyai.com/claude-code-resource-list/
- hesreallyhim. Awesome Claude Code. https://github.com/hesreallyhim/awesome-claude-code
- ComposioHQ. Awesome Claude Skills. https://github.com/ComposioHQ/awesome-claude-skills
- Jeremy Longshore. Claude Code Plugins + Skills (270+ plugins). https://github.com/jeremylongshore/claude-code-plugins-plus-skills



© 2026 KENNETH BAREKSTEN

Denne boka er lisensiert under Creative Commons (CC BY 4.0). Det betyr at du kan dele, bruke og tilpasse innholdet, også kommersielt, så lenge du krediterer forfatteren.



Kontakt: kenneth@laererliv.no
Nettside: laererliv.no