

# Outline

- **Løkker**
- Kombinere løkker og samlinger
- For-løkker
- Prosedyrer med parametre
- Funksjoner (returverdier)

# Repetert kjøring med prosedyrer

- Om vi ønsker å repetere kjøring kan vi:
  - Legge funksjonaliteten i en navngitt kodeblokk (prosedyre)
  - Kalle denne prosedyren flere ganger
  - {tre\_velkomster.py}
- Vi er imidlertid bundet til et fast antall kjøringer (tilsvarende antall kall)
  - For å kjøre et fleksibelt antall ganger trenger vi en løkke

# Repetert kjøring (løkke): **while**

- Syntaks:
  - while condition:  
    Statement  
    Statement
- Eksempel:
  - tall=1  
  while tall<100:  
    print(tall)  
    tall+=5
- En slags if med tilbakekobling:
  - Nesten som if, bare at man kjører innholdet mange ganger - helt til condition ikke lenger er True

# Et eksempel på repetert kjøring

- {matte\_test.py}

# While som en if med tilbakekobling

```
innlest = input("Hva er 4+7?")  
tall = int(innlest)
```

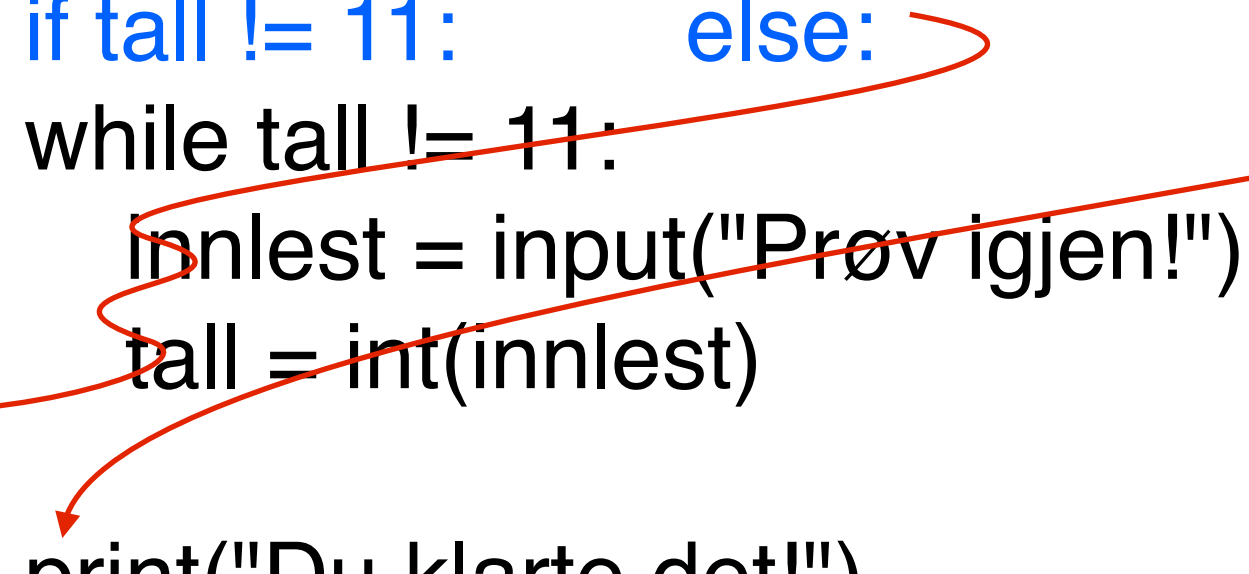
```
while tall != 11:  
    innlest = input("Prøv igjen!")  
    tall = int(innlest)
```

```
print("Du klarte det!")
```

# While som en if med tilbakekobling

```
innlest = input("Hva er 4+7?")  
tall = int(innlest)
```

```
if tall != 11:      else:  
    while tall != 11:  
        innlest = input("Prøv igjen!")  
        tall = int(innlest)  
print("Du klarte det!")
```



The diagram consists of red arrows illustrating the flow of execution. One arrow starts from the 'else:' branch of the 'if' statement and points to the 'while' loop, indicating that the while loop is executed when the condition is false. Another arrow starts from the end of the 'while' loop and points back to the 'while' loop's condition, representing the loop's back-coupling mechanism.

# Eksempelet vi startet med

- Repetert spørring frem til brukeren oppgir negativ alder:
  - {velkomst\_lokke\_feil.py, velkomst\_lokke\_uperfekt.py}

# Merk den presise rekkefølgen ting blir gjort!

- while condition:  
    statement1  
    statement2 ...
- Man sjekker,  
    kjører hele blokka,  
    og går så tilbake til sjekken igjen
- Sjekk condition,  
    statement1,  
    statement2,  
    sjekk condition igjen,  
    statement1  
    statement2 ..



# Merk den presise rekkefølgen ting blir gjort!

- while condition:  
    statement1  
    statement2 ...
- Det blir altså **ikke** sjekket noe  
  mellom statement1 og statement2
  - Det som sjekkes er oppfylt når man starter å kjøre  
  kodeblokken
  - .. men det kan slutte å være oppfylt underveis i blokken

# Finjustering av når noe sjekkes og brukes

- {velkomst\_lokke\_fungerer.py}: Innlesning av alder lagt sist i løkka, slik at verdi sjekkes like etter innlesning
- Unngår å skrive alders-basert kommentar etter terminerende input (-1) fra bruker
- Man må legge en ekstra linje med innlesning før selve løkka begynner

# En liten oppgave

- Skriv kode som regner ut summen av tallene fra 1 til 100  
( $1+2+3...+100$ )
- Prøv selv med blyant og papir!
- Etterpå diskuter med nabo
- {sum\_vha\_while.py}

# Noen lærdommer å hente fra oppgaven/løsningen

- Løkker lar oss splitte opp en problemstilling
  - Oppgaven spør om mange verdier (hundre tall)
  - Poenget med løkken er å kunne behandle ett tall om gangen
  - Første steg i å løse problemet er å sørge for at hvert tall vi trenger oppstår én gang i løkken
- Vi må også sørge for å få slått sammen hver bit til en fullstendig løsning
  - I dette tilfellet la vi til hver bit i variabelen summen
  - Andre steg er altså å finne ut hvordan slå sammen bitene

# Eksempel på bruk av løkke: Monte Carlo-simulering

- Problemstilling:  
dersom man triller to terninger, hva er sannsynligheten for å tilsammen få 10 eller mer?
- Man kan lære seg sannsynlighetsregning
- Eller man kan la datamaskinen trille terning og bare telle hvor ofte summen blir 10 eller mer
- {terning.py}

# Monte Carlo-simulering

- Monte Carlo-simulering lar oss løse vanskelige beregnings-problemer med enkel matematikk
  - I stedet for å benytte avanserte matematiske formler, kan man ofte bare la datamaskinen etterligne det man er interessert i (simulere),  
og så ta et enkelt gjennomsnitt av utfallene
- Typisk fremgangsmåte i Python:
  - Ha en løkke som simulerer veldig mange ganger
  - Hver gang gjøre noe tilfeldig trekk (bruk *random*)
  - Inni løkken telle opp det man er interessert i, og så ta et gjennomsnitt etterpå (f.eks. antall ganger noe slo til)

# Et ekte problem

(på en travel morgen)

- Problemstilling:
  - Jeg har en kopp hvor det ligger 4 løse kontaktlinser, 2 for venstre øye og 2 for høyre øye
  - Av ren latskap grabber jeg to vilkårlige linser fra koppen (av de 4) og håper at jeg har grabbet et for venstre og et for høyre (ellers må jeg lete videre).
- Hvor ofte blir det riktig (én for hvert øye)?
  - Vanligvis riktig (over 50% sjanse for å få én for hver)?
  - Vanligvis feil (mindre enn 50% sjanse for én for hver)?
  - Riktig annenhver gang (nøyaktig 50% sjanse)?