Hvordan løse problemer med programmering?

Problemløsning, løkker, og kombinering av løkker og samlinger

IN1000, uke4
Geir Kjetil Sandve

Outline

- Løkker
- Kombinere løkker og samlinger
- For-løkker

Outline

- **Løkker**
- Kombinere løkker og samlinger
- For-løkker

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!")
```

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 kodeblokka
    - .. men det kan slutte å være oppfylt underveis i blokka

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)?
- {linser.py}

Outline

- Løkker
- **Kombinere løkker og samlinger**
- For-løkker

Løkker og samlinger

- Løkker og lister arbeider ofte godt sammen!
  - Løkker gjør at kodelinjer kan kjøres flere ganger
  - Lister gjør det mulig å jobbe med mange verdier
- {huske1-2.py}

Løkker og samlinger

- Typisk bruk av løkke og liste:
  - En variabel holder en liste av en gitt lengde
  - En annen variabel holder en indeks for listen, starter på 0
  - En while-løkke øker indeksen med én for hver iterasjon, inntil indeksen er utenfor listen
  - For hver iterasjon i løkken henter eller endrer man liste-verdien på den aktuelle indeksen
- Andre vanlig bruksmåter
  - For hver iterasjon av løkken, sammenligne verdien på den aktuelle indeksen med verdien på indeksen før eller etter
  - Kjøre en løkke et bestemt antall ganger. Begynne med en tom liste og legge til ett nytt element for hver iterasjon

Oppgave

(Oppg 3b fra eksamen 2016, lettere omskrevet)

- Skriv kode som sjekker om alle verdiene i en liste av heltall er ekte større enn 10 og ekte mindre enn 20. Dersom alle verdiene er innenfor dette intervallet skal en variabel *innenfor* settes til verdien True, ellers skal *innenfor* settes til False.

```
tallene = [12, 16, 5, 16]
#Skriv din kode her
assert innenfor==False
```

En mulig løsning

- {innenfor.py}

Outline

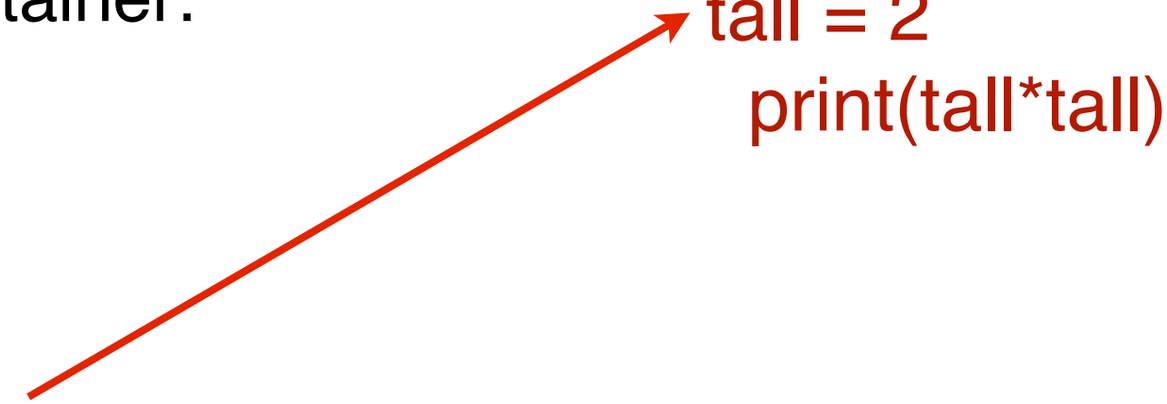
- Løkker
- Kombinere løkker og samlinger
- **For-løkker**

Iterere gjennom en samling: **for**

- Syntaks:
  - for variable in collection:
    statement1
    statement2
    ...
- Eksempel:
  - for tall in [2,3,4]:
    print(tall*tall)
- En løkke som kjøres én gang med hver verdi i en samling (collection)
  - Variabelen mellom "for" og "in" blir satt til én verdi fra samlingen for hver gang kodeblokken i løkka kjøres

Iterere gjennom en samling: **for**

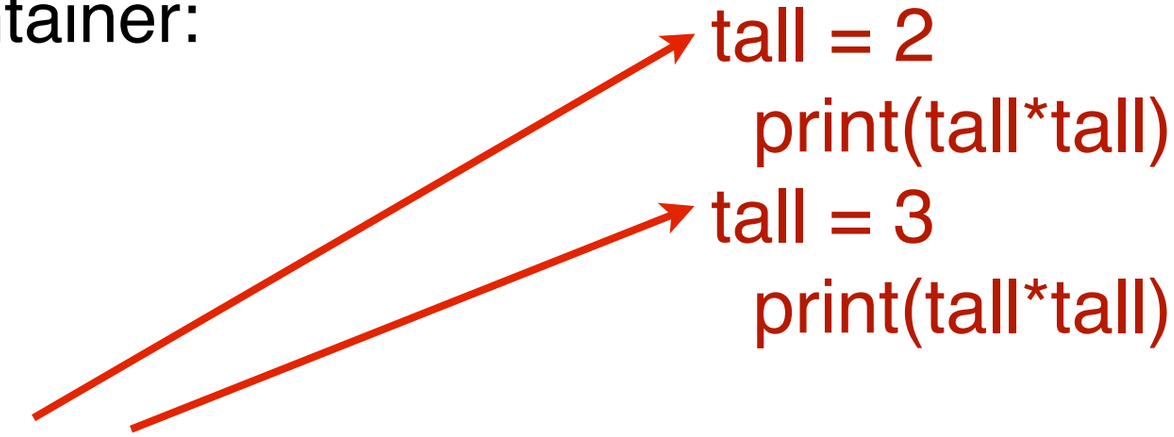
- Syntaks:
  - for variable in container:
    statement1
    ...
- Eksempel:
  - for tall in [2,3,4]:
    print(tall*tall)



- En løkke som kjøres én gang med hver verdi i en samling (container)
  - Variabelen mellom "for" og "in" blir satt til én verdi fra samlingen for hver gang kodeblokken i løkka kjøres

Iterere gjennom en samling: **for**

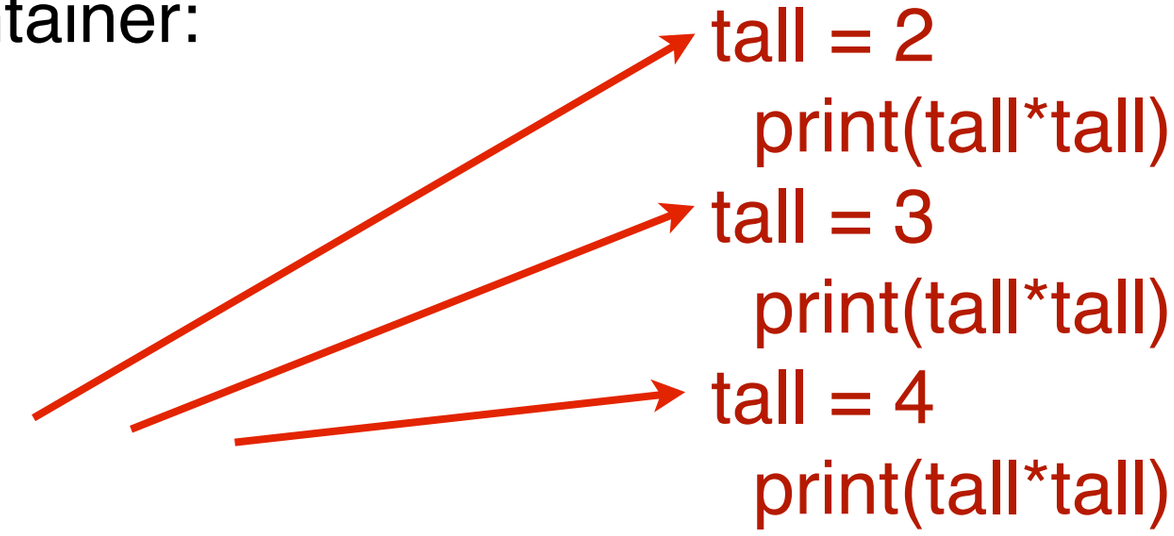
- Syntaks:
  - for variable in container:
    statement1
    ...
- Eksempel:
  - for tall in [2,3,4]:
    print(tall*tall)



- En løkke som kjøres én gang med hver verdi i en samling (container)
  - Variabelen mellom "for" og "in" blir satt til én verdi fra samlingen for hver gang kodeblokken i løkka kjøres

Iterere gjennom en samling: **for**

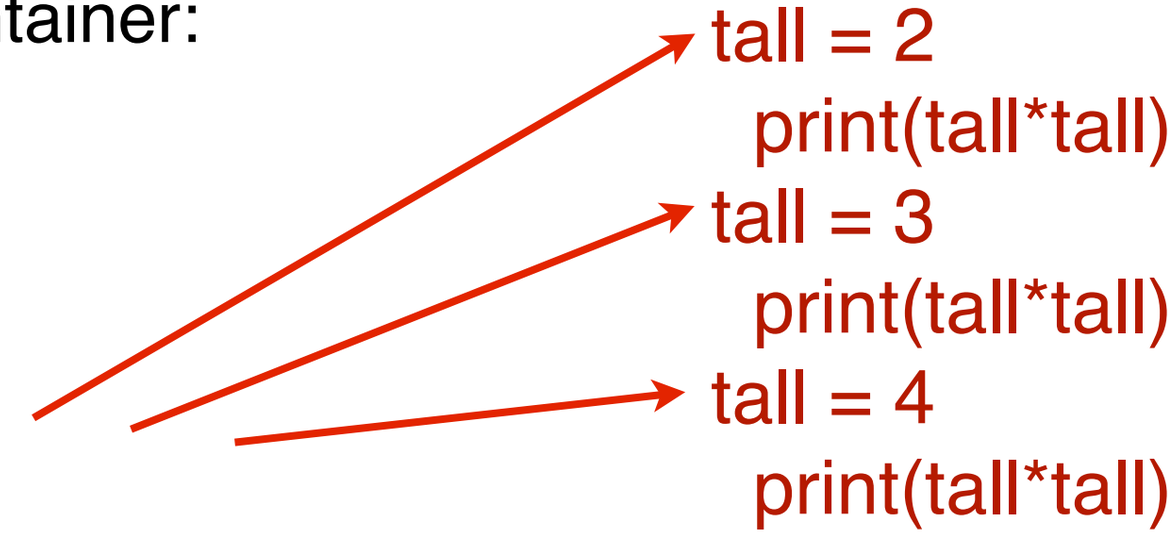
- Syntaks:
  - for variable in container:
    statement1
    ...
- Eksempel:
  - for tall in [2,3,4]:
    print(tall*tall)



- En løkke som kjøres én gang med hver verdi i en samling (container)
  - Variabelen mellom "for" og "in" blir satt til én verdi fra samlingen for hver gang kodeblokken i løkka kjøres

Iterere gjennom en samling: **for**

- Syntaks:
  - for variable in container:
    statement1
    ...
- Eksempel:
  - for tall in [2,3,4]:
    print(tall*tall)



- En løkke som kjøres én gang med hver verdi i en samling (container)
  - Variabelen mellom "for" og "in" blir satt til én verdi fra samlingen for hver gang kodeblokken i løkka kjøres
- [sum_vha_for_v1.py]

Samme oppgave igjen:

Kan du nå løse den med for-løkke?

- *Skriv kode som sjekker om alle verdiene i en liste av heltall er ekte større enn 10 og ekte mindre enn 20. Dersom alle verdiene er innenfor dette intervallet skal en variabel innenfor settes til verdien True, ellers skal samme variabel settes til False.*

```
tallene = [12, 16, 5, 16]
#Skriv din kode her
assert innenfor==False
```

En mulig løsning

- {innenfor2.py}

Kjapt spesifisere lister av tall: *range*

- Funksjonen *range* kan brukes for å kjapt lage lister:
  - print(range(5)) # range(0, 5)
  - print(list(range(5))) # [0,1,2,3,4]
  - range(2,5) # [2,3,4]
  - range(0,5,2) # [0,2,4]
  - range(1,5,2) # [1,3]
- Dette kan brukes (direkte) som samling i en for-løkke
  - for tall in range(1,5):
    print(tall*tall)
- {sum_vha_for_v2.py}

For-løkke og indekser

For-løkke og indekser

-
  - taxi_kostnader = [145, 220, 91, 340]
    for kostnad in taxi_kostnader:
      if kostnad<100:
        kostnad = 100

For-løkke og indekser

- For å sette verdier i listen må man bruke indeks(!)
  - taxi_kostnader = [145, 220, 91, 340]
    for kostnad in taxi_kostnader:
      if kostnad<100:
        kostnad = 100
    print(taxi_kostnader)



NB! Endrer kun variabelen kostnad - ikke listen

[145, 220, **91**, 340]

For-løkke og indekser

- For å sette verdier i listen må man bruke indeks(!)

  - taxi_kostnader = [145, 220, 91, 340]
    for kostnad in taxi_kostnader:
      if kostnad<100:
        kostnad = 100
    print(taxi_kostnader)

✗ NB! Endrer kun variabelen kostnad - ikke listen

[145, 220, **91**, 340]

  - taxi_kostnader = [145, 220, 91, 340]
    indeks = 0
    while indeks < len(taxi_kostnader) :
      if taxi_kostnader[indeks] <100:
        taxi_kostnader[indeks] = 100
      indeks += 1
    print(taxi_kostnader)

✓ Endrer listen

[145, 220, **100**, 340]

Siden strenger er en type lister

- Man kan iterere gjennom strenger:
  - for bokstav in "NORGE":
    print("Gi meg en " + bokstav)
- Man kan også aksessere strenger på indeks:
  - land = "NORGE"
    for indeks in range(len(land)):
    print("Gi meg en " + land[indeks])

Oppsummering

- Løkker gjør at kodelinjer kan kjøres flere ganger
  - F.eks. gjøre lignende type utregning på ulike verdier
- Løkker og lister jobber godt sammen
  - Kan generere og oppsummere store mengder verdier
- Dere har nå verktøykassen for å løse skikkelige problemstillinger!