Løkker og lister

Løse problemer med programmering

INF1001, uke3
Geir Kjetil Sandve

Hva vi har lært så langt

- Variabler og uttrykk
- Beslutninger
- Kontrollflyt
- Prosedyrer

Fokus i dag

- Repetert kjøring (løkker)
- Holde på mange verdier (lister og mere)
- Løse problemer vha programmering

Løkker

Repetert kjøring med prosedyrer

- Om vi ønsker å repetere kjøring kan vi:
  - Legge funksjonaliteten i en navngitt kodeblokk (prosedyre)
  - Kalle denne prosdyren 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

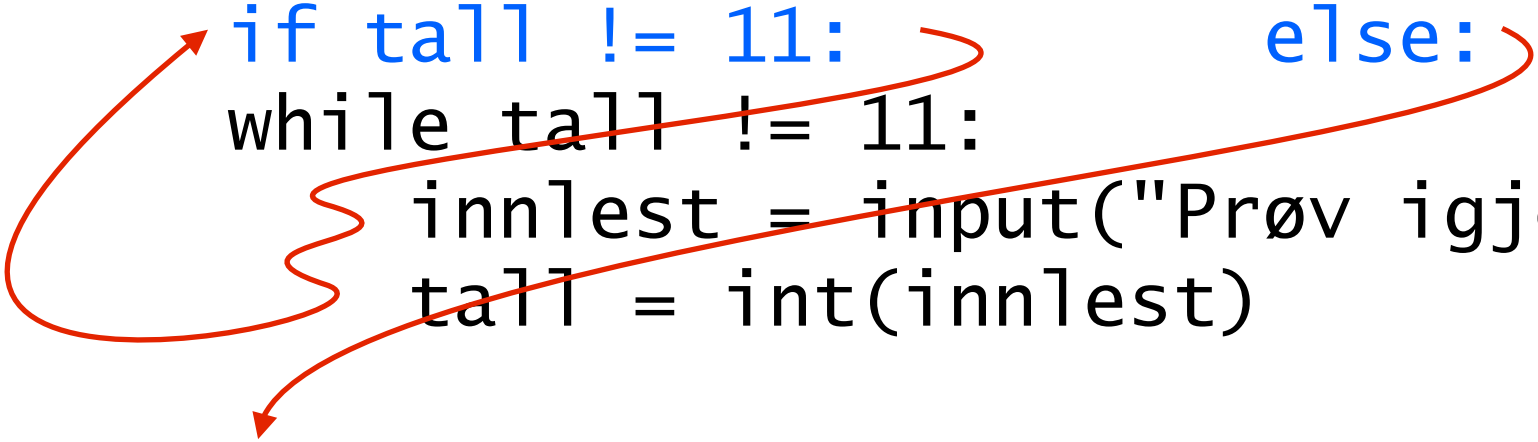
Et eksempel på repetert kjøring

- {matte_test.py}

While som en **if** med tilbakekobling

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

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

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

else:



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 (det essensielle) som regner ut summen av tallene fra 1 til 100 *(1+2+3...+100)*
  - Sett på pause og prøv selv med blyant og papir
- {sum_vha_while.py}

Lister

Sjonglere med flere verdier

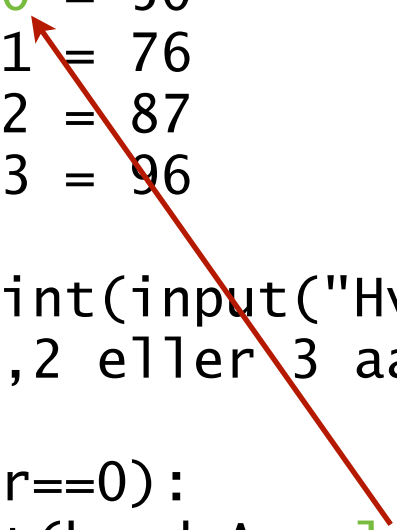
- {hoyde1.py}

Finne verdien vi trenger direkte

```
hoydeAar0 = 50
hoydeAar1 = 76
hoydeAar2 = 87
hoydeAar3 = 96

alder = int(input("Hvilken alder vil du vite hoyden
for (0,1,2 eller 3 aar)? "))

if (alder==0):
    print(hoydeAar0)
elif (alder==1):
    print(hoydeAar1)
elif (alder==2):
    print(hoydeAar2)
elif (alder==3):
    print(hoydeAar3)
```

Finne verdien vi trenger direkte

```
hoydeAar0 = 50
hoydeAar1 = 76
hoydeAar2 = 87
hoydeAar3 = 96

alder = int(input("Hvilken alder vil du vite hoyden
for (0,1,2 eller 3 aar)? "))

if (alder==0):
    print(hoydeAaralder)
elif (alder==1):
    print(hoydeAar1)
elif (alder==2):
    print(hoydeAar2)
elif (alder==3):
    print(hoydeAar3)
```



Vi kan slå opp verdien vi trenger direkte!

- Det vi ønsket:
  - hoydeAar**alder**
- Syntaks i Python:
  - hoydeAar[alder]
- Og før dette må vi definere hoydeAar som en liste:
  - hoydeAar = [50, 76, 87, 96]

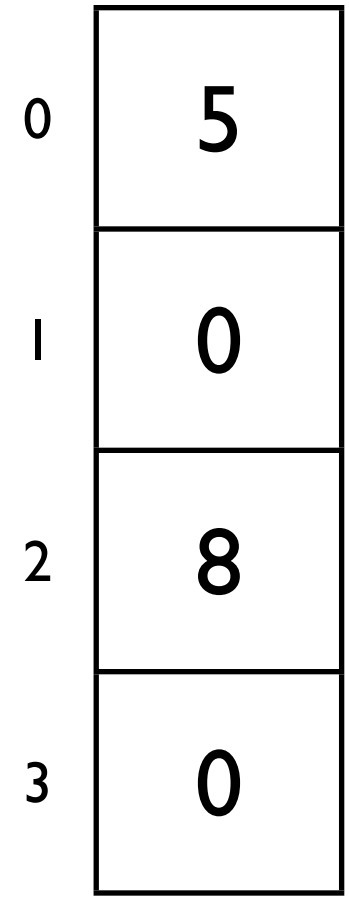
Håndtere høydene i en array

- {hoyde2.py}

Liste

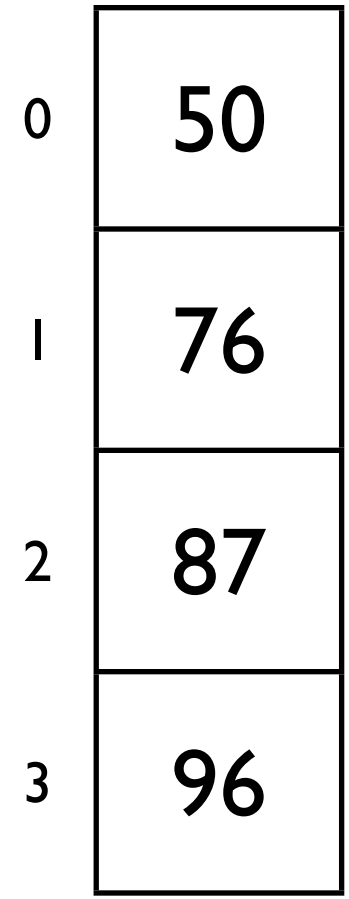
- Definere en liste:
  - hoydeAar = [50, 76, 87, 96]
  - hoydeAar = []
  - hoydeAar = [0] * 4
- Sette en enkeltverdi:
  - hoydeAar[0] = 5
  - hoydeAar[2] = 8
- Bruke enkeltverdi
  - print(hoydeAar[0])

5



Bygge en liste

- Først definere en tom liste
  - hoydeAar = []
- Utvide med en enkeltverdi:
  - hoydeAar.append(50)
  - hoydeAar.append(76)
- Konkatenere lister
  - print(hoydeAar + [87,96])
  - hoydeAar = hoydeAar + [87,96]



Liste - funksjonalitet

- Kan inneholde alle typer verdier
  - min_liste = [1.5, 2.9, 1.0]
  - min_liste = ["Oslo", "Bergen"]
- Lengde av liste:
  - len(min_liste) 2
- Sjekke om en verdi finnes:
  - "Bergen" in min_liste True
  - "Trondheim" in min_liste False

En liten test

```
vest = ["Hallo", "Bergen"]
midt = ["Trondheim"]
print( vest + midt) ['Hallo', 'Bergen', 'Trondheim']
nord = ["Alta", "Kautokeino"]
vest = nord + vest
print(vest)  ['Alta', 'Kautokeino', 'Hallo', 'Bergen']
nord.append("Narvik")
print(nord)  ['Alta', 'Kautokeino', 'Narvik']
lengde = len(vest+nord)
print(lengde) 7
```

Løkker og lister

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

En (utfordrende) oppgave

(Oppg 5 fra eksamen 2014, lettere omskrevet)

- Skriv et program som sjekker om alle verdiene i en liste ***min_liste*** er i stigende rekkefølge (sortert). Dersom alle verdiene er i sortert rekkefølge skal en variabel ***er_sortert*** settes til *True*, ellers skal denne settes til *False*.

```
min_liste = [1,2,4,3,5]
#Skriv din kode her
assert er_sortert==False
```

Lister, mengder og ordbøker

Mengder

- Fagene man tar et semester
  - Har ingen spesifikk rekkefølge (hva er fag 1, 2 og 3..)
  - Alle fag må være ulike (kan ikke ta INF1001 og INF1001)
- Kan representeres med en liste, men:
  - Får uansett en rekkefølge (som dog kan ignoreres): ["INF1001","INF1080","EXPHIL03"]
  - Ingenting hindrer å ha samme fag to ganger: ["INF1001","INF1001","INF1001"]

Mengder

- Mengde:
  - En samling av ulike verdier
  - Det vil si: a) uten ordning, b) kun ulike verdier
- Mengde i Python:
  - min_mengde = set([1,5,1,1])
  - len(min_mengde) 2
  - 5 in min_mengde True
  - ~~min_mengde[1]~~

Ordbøker
(dictionaries)

- Tilbake til listen hoydeAar: [50, 76, 87, 96]
  - En samling av fire ulike høyder
  - En slags ordbok fra alder til høyde:
    0->50, 1->76, 2->87, 3->96
- Hva om man vil legge til én ekstra sammenheng?
  - 18->179
  - Hva gjør man med indeksene mellom 3 og 18?
  - [50, 76, 87, 96, 0, 0, 0,0,0,0,0,0,0,0,0,0, 0, 0, 179] **??**
- Man bruker en ordbok (dict)
  - {0:50, 1:76, 2:87, 3:96, 18:179}

Ordbøker

- En ordbok (dict) er en samling mappinger (transformasjoner) fra én verdi til én annen
  - Det man mapper fra kalles nøkkel (key)
  - Det man mapper til kalles en verdi (value)
- Både nøkler og verdier kan være av ulike typer
  - by = {"Norge":"Oslo", "Tyskland":"Berlin", "Italia":"Roma"}
  - tlf = {"Norge": 47, "Tyskland": 49, "Italia": 39}
- Slår opp som i en liste:
  - by["Norge"] Oslo

Ordbok - funksjonalitet

- Lage en liste
  - min_ordbok = {}
  - min_ordbok = {"Norge": 47, "Tyskland": 49}
- Legge til ekstra element
  - min_ordbok["Italia"] = 39
- Slå opp i ordlisten
  - print(min_ordbok["Tyskland"] - min_ordbok["Norge"]) 10
- Sjekke hvilke *nøkler* den inneholder
  - "Norge" in min_ordbok True
  - 47 in min_ordbok False *(sjekker nøklene, ikke verdiene!)*

Oppsummering

- Løkker gjør at kodelinjer kan kjøres flere ganger
  - F.eks. gjøre lignende type utregning på ulike verdier
- Lister gjør det mulig å jobbe med mange verdier
  - Kan slå opp direkte på verdien i en bestemt posisjon
- Løkker og lister jobber godt sammen
  - Kan generere og oppsummere store mengder verdier
- Man har flere typer samlinger til disposisjon:
  - Lister, mengder, ordbøker
- Dere har nå verktøykassen for å løse skikkelige problemstillinger!