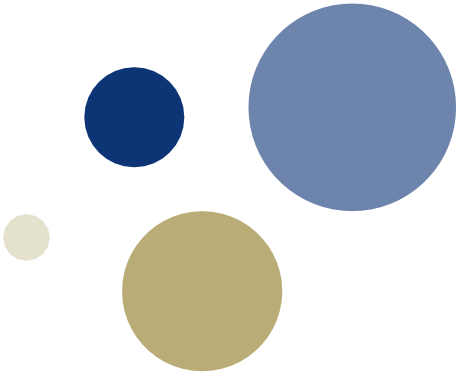
NTNU

Kunnskap for en bedre verden



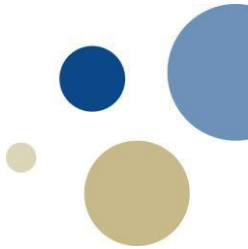
Python: Løkker (while og for)

TDT4110 IT Grunnkurs

Dag Olav Kjellemo

Prof Guttorm Sindre

Læringsutbytter denne uka



- Kunne forklare
  - Nytteverdien til løkker
  - Virkemåten til kode med løkker
    - Sammensatte tilordningsoperatorer
    - Betingelsesstyrte løkker - **while** i Python
    - Sekvensstyrte løkker – **for** i Python
- Kunne løse problemer ved hjelp av løkker
  - Velge riktig løkkestruktur avhengig av behov
    - **while** eller **for** ?
    - Ingen løkke? Enkel? Dobbel?
  - Skrive fungerende kode med løkker

Nytteverdien til løkker

- Kunne repetere kodelinjer mange ganger
  - Effektiv repetisjon er en av datamaskinens styrker
- Hvorfor ikke heller kopiere kodelinjene i ei lang remse?
  - Upraktisk: Koden lang og uoversiktlig
  - Av og til umulig:
    - Hva om vi har et ukjent / fleksibelt antall repetisjoner?
- To typer av løkker:
  - **while**: passer best til ukjent antall repetisjoner
  - **for**: passer best til kjent antall, gå gjennom en sekvens av data

Løkker - gjenta kodelinjer flere ganger

Det er nyttig å kunne gjenta en del av et program flere ganger, uten å skrive det samme mange ganger.

Til dette bruker man løkker.

Dette er tungvint, bruk heller løkke:

```
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
print('Løkker er lurt!')
…
```

Ordet *løkke* brukes fordi vi skal gjenta noe flere ganger (rundt og rundt)

Med løkke:

```
for i in range(10):
    print('Løkker er lurt!')
```

While-løkker:

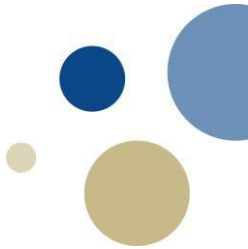
Ligner litt på if-setning, men True-blokken gjentas helt til betingelsen er False.

```
# En nyttig struktur som ikke gjør noe nyttig, ennå:

fortsett = True

while fortsett:
    linje = input('> ')
    fortsett = linje == ''
```

For hver av



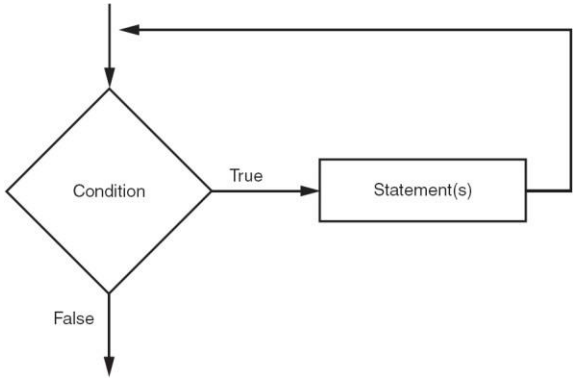
To typer av løkker:

- **while**: passer best når det er en betingelse / test som bestemmer om vi skal fortsette eller slutte løkka.

- **for**: passer best når vi har et gitt/kjent antall ganger vi skal gjenta løkka, eller at vi har et kjent/gitt sett med data, som skal behandles I løkka.

Oppsett av while-løkke

```
while betingelse:
    kodelinje1
    kodelinje2
kodelinje3
```



- Minner om **if**
  - Kode med innrykk hører til løkka
    - hvis betingelse er **True** blir linjer med innrykk utført
    - hvis betingelse er **False** blir de ikke utført
  - kodelinje3 gjøres etter løkka, uansett
- I motsetning til **if**
  - kodelinje1 og 2 ***gjentas*** inntil betingelse blir **False**
    - null eller flere ganger
  - Hvis betingelsen aldri blir **False**: ”evig løkke”

Se notebook FP_30_while
(lenke på Blackboard)

En liten digresjon: Sammensatte tilordningsoperatorer

Forekommer ofte i løkker

Vanlig	Sammensatt	Hva gjøres
`x = x + 4`	`x += 4`	Øker verdien til x med 4
`x = x - 3`	`x -= 3`	Minsker verdien til x med 3
`x = x * 10`	`x *= 10`	Tidobler verdien til x
`x = x / 2`	`x /= 2`	Halverer verdien til x
`x = x // 2`	`x //= 2`	Halverer verdien til x (heltallsresultat)
`x = x % 3`	`x %= 3`	Ny verdi for x blir gml verdi modulo 3

Merk: Vanlig og kompakt form er

- ekvivalente for **immuterbare** datatyper (f.eks. int, float, str)
- **ikke** ekvivalente for **muterbare** datatyper (f.eks. list)
  - (skal forklare dette når vi kommer til lister i pensum)

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Menti: hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

Se notebook FP_32_...virkemåte
(lenke på Blackboard)

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

n = 0	Før	
summen	0	
tall	1	
tall <= n	**False**	Løkka blir **ikke** kjørt
return	0	

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

n = 5	Før	Runde1
summen	0	0+1 -> 1
tall	1	1+1 -> 2
tall <= n	True	True
return		

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

n = 5	Før	Runde1	Runde2
summen	0	0+1 -> 1	1+2 -> 3
tall	1	1+1 -> 2	2+1 -> 3
tall <= n	True	True	True
return			

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

n = 5		R1	R2	R3
summen	0	1	3	6
tall	1	2	3	4
tall <= n	True	True	True	True
return				

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

n = 5		R1	R2	R3	R4
summen	0	1	3	6	10
tall	1	2	3	4	5
tall <= n	True	True	True	True	True
return					

Virkemåte while-løkke

- Kjører nye runder så lenge betingelsen er **True**
- Når False
  - Utfører IKKE koden i løkka
  - Går videre med koden under løkka (dvs. setning(er) som ikke er på innrykk)
- Hva returneres fra **sum_1_n(5)** og **sum_1_n(-1)?**

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

n = 5		R1	R2	R3	R4	R5
summen	0	1	3	6	10	15
tall	1	2	3	4	5	6
tall <= n	True	True	True	True	True	False
return						15

for-løkker

- Gjenta ei kodeblokk *et kjent / begrenset antall ganger*
- Generelt format:

  for variabel in iterable:

  kodelinjer med innrykk

- *iterable* er et objekt som inneholder en *sekvens,* f.eks.
  - Liste
    - Gitt direkte, f.eks. [0, 1, 2, 3] ; [’♥’, ‘♠’, ‘♦’, ‘♣’] ; [‘A’, 2, 3, 4, 5, 6, 7, 8, 9, 10, ‘J’, ‘Q’, ‘K’]
    - Eller i en variabel, f.eks. **kortfarger** eller **kortverdier**
  - Numpy-array
  - En tekststreng
  - Et range()-objekt

Se notebook FP_31_for
(lenke på Blackboard)

Bruk av range-objekter

- Et range-objekt husker en sekvens av tall
  - Lagrer **ikke** alle tallene
    - bare startverdi, sluttverdi og stegverdi
    - sparer plass for lange tallrekker
  - Opprettes ved range()
  - range() kan ta ett, to eller tre argumenter
    - NB: alle må være heltall (int)
    - **numpy.arange()** kan også ta desimaltall, returnerer et array
- Illustrasjon:

```
range(8)        # fra: 0, til: 8, sekvensen blir 0, 1, 2, 3, 4, 5, 6, 7
range(2, 8)     # sekvensen blir 2, 3, 4, 5, 6, 7
range(3, 20, 4) # sekvensen blir 3, 7, 11, 15, 19
range(21, 5, -3) # sekvensen blir 21, 18, 15, 12, 9, 6
```

Se notebook FP_31_for
(lenke på Blackboard)

Hvilken range() gir hvilken tallsekvens?

A range(5)

B range(4)

C range(3, 12, 2)

D range(1, 4, 3)

E range(9, 0, -3)

p	0, 1, 2, 3
q	0, 1, 2, 3, 4, 5
r	0, 1, 2, 3, 4
s	3, 5, 7, 9, 11
t	3, 5, 7, 9, 12
u	1
v	1, 4
w	1, 4, 3
x	9, 6, 3, 0
y	9, 6, 3, 0, -3
z	9, 6, 3

Gi inn 5 svar, hvert med to bokstaver.
F.eks. **Az** hvis du tror range **A** til venstre gir sekvens z til høyre

Virkemåte for-løkke

Se notebook FP_32_...virkemåte
(lenke på Blackboard)

- variabel får verdier…
  - Første runde: variabel = første element i sekvensen
  - Andre runde: variabel = andre element i sekvensen
  - …
  - Siste runde: variabel = siste element i sekvensen
- Hvis sekvensen er tom hopper vi forbi løkka uten å utføre den
- Denne funksjonen gir samme resultat som sum_1_n_while()

```
def sum_1_n_for(n):
    summen = 0
    for tall in range(1, n+1):
        summen += tall
    return summen
```

Virkemåte for-løkke

Se notebook FP_32_...virkemåte
(lenke på Blackboard)

- variabel får verdier…
  - Første runde: variabel = første element i sekvensen
  - Andre runde: variabel = andre element i sekvensen
  - …
  - Siste runde: variabel = siste element i sekvensen
- Hvis sekvensen er tom hopper vi forbi løkka uten å utføre den
- Denne funksjonen gir samme resultat som sum_1_n_while()

```
def sum_1_n_for(n):
    summen = 0
    for tall in range(1, n+1):
        summen += tall
    return summen
```

n = 0	Før	
summen	0	
tall		
range(1,1)	**tom**	Løkka blir **ikke** kjørt
return	0	

Virkemåte for-løkke

Se notebook FP_32_...virkemåte
(lenke på Blackboard)

- variabel får verdier…
  - Første runde: variabel = første element i sekvensen
  - Andre runde: variabel = andre element i sekvensen
  - …
  - Siste runde: variabel = siste element i sekvensen
- Hvis sekvensen er tom hopper vi forbi løkka uten å utføre den
- Denne funksjonen gir samme resultat som sum_1_n_while()

```
def sum_1_n_for(n):
    summen = 0
    for tall in range(1, n+1):
        summen += tall
    return summen
```

n = 5		R1	R2	R3	R4	R5
summen	0	1	3	6	10	15
tall		1	2	3	4	5
range(1,6)	1,2,3,4,5	~~1,~~2,3,4,5	~~1,2,~~3,4,5	~~1,2,3,~~4,5	~~1,2,3,4,~~5	~~1,2,3,4,5~~
return						15

Løse problemer med løkker

- Mange problemer kan løses både med while og for
- Eller UTEN bruk av løkke
  - Selv om problemet «ser ut» som et typisk løkkeproblem

- Eksempel 1: Sum fra 1-N, bruke løkke eller ikke?
- Eksempel 2: Tverrsummen av et tall
  - Versjon med **while**, versjon med **for**

- Eksempel 3: Plotting
  - Når kan vi slippe løkke (pga samleoperasjoner på numpy arrays)?
  - Når kan løkke være nødvendig eller nyttig?

Sum 1-N, løkke eller ikke?

- Fem alternative løsninger, hvilken tror du er best?
  - Må funke greit både for heltall >0, både små og store
  - Viktig: riktig svar, hastighet, strømsparing

```
def sum_1_n(n):
    summen = 0
    tall = 1
    while tall <= n:
        summen += tall
        tall += 1
    return summen
```

```
def sum_1_n_range(n):
    return sum(range(1,n+1))

def sum_1_n_arange(n):
    return np.sum(np.arange(1,n+1))

def sum_1_n_gauss(n):
    return (n+1) * n // 2
```

```
def sum_1_n_for(n):
    summen = 0
    for tall in range(1, n+1):
        summen += tall
    return summen
```

Se notebook BP_33_løkker_tallserier
(lenke på Blackboard)

Typisk bruk av ulike løkker

- for-løkke: kjent antall repetisjoner
  - fast eller kjent antall når løkka starter
  - Eksempel: samme operasjon på…
    - alle elementer i en tabell eller liste
    - alle tall i et intervall eller en tallrekke
- while-løkke: også ukjent antall repetisjoner, f.eks
  - inntil brukeren ønsker å avslutte
  - inntil et mål er nådd, f.eks.:
    - Beregninger: Til svaret er nøyaktig nok
    - Kontroll/styringssystemer: Til en ønsket tilstand er nådd
    - Søking: Til ønskede data er funnet
    - Spill: Til noen har vunnet
    - …

Hvordan tenke med løkker?

- Gjøre **før** løkka? F.eks.
  - Lese inn data fra bruker?
  - Gi startverdier til variable som brukes inni løkka
  - Gjøre det vi *kan* av utregninger
    - minst mulig jobb inni løkka
- Hva må vi gjøre inni løkka?
  - Hvilke handlinger skal gjentas for hver runde?
  - **while**: sørge for at vi nærmer oss sluttbetingelsen
- Hva gjøre etter løkka?
  - Ta vare på resultatet for å bruke det videre
  - (Evt. i små kodeeksempler: printe ut)

Eksempeloppgave

- Lag en funksjon tverrsum(n) som
  - Får inn et positivt heltall n
  - Returnerer tverrsummen til n
    - F.eks. tverrsum(125) skal bli 8, fordi 1+2+5 blir 8
- Skal virke for vilkårlig store heltall

- Ser på to løsninger:
  - Matematisk inspirert løsning
  - Løsning basert på typekonvertering

Se notebook BP_34_tverrsum
(lenke på Blackboard)

Matematisk inspirert løsning

- Idé: Finne ett og ett siffer, plusse inn i summen etter hvert
  - Hvordan finne fremste siffer? Ikke trivielt…
  - Bakerste siffer? Lettere, det er **n % 10**
    - Hvordan deretter finne nest bakerste?
    - n // 10 gir oss tallet uten det bakerste sifferet
  - Og deretter gjenta trikset med n % 10
- En mulig innmat for løkka:

```
siffer = n % 10
summen += siffer
n = n // 10
```

- Hva må gjøres før løkka:
  - Gi startverdi til summen (0)
- Etter løkka? Returnere resultat
- Hvor lenge skal løkka holde på?
  - Så lenge n > 0, har vi fortsatt siffer vi kan plusse inn

```
def tverrsum_while(n):
    summen = 0
    while n > 0:
        siffer = n % 10
        summen += siffer
        n = n // 10
    return summen
```

Løsning med typekonvertering

- Hvorfor ikke bare kjøre ei for-løkke gjennom alle sifrene?
  - Fordi et heltall **ikke** er en sekvens
- Men kan vi konvertere til noe som er en sekvens?
  - Ja! Til streng, med str(n)
- Denne strengen kan gjennomløpes med ei for-løkke
  - Første runde: første siffer, andre runde: andre siffer, osv.
- Konvertere hvert siffer til tall med int() og addere
- Før løkka, initiere summen (0), etter returnere (som for while)

```
def tverrsum_for(n):
    summen = 0
    for siffer in str(n):
        summen += int(siffer)
    return summen
```

Eks., løkke med numpy array

- Vi har tidligere (bl.a. plotting i Ø1) sett at
  - Vi trenger ikke løkker for å regne y-verdier for en masse x-verdier i et numpy array
  - Kan i stedet kjøre hele arrayet gjennom beregningen
- Men dette forutsetter at
  - Funksjonen vi skal beregne bruker vanlige operatorer
  - Eller numpy-funksjoner
- Hvis funksjonen f.eks. inneholder if-setning…
  - Ikke så enkelt
  - Se notebook BP_35 om plotting…

Se notebook BP_35_plotting
(lenke på Blackboard)

Oppsummering

- **while**-løkke: en betingelse avgjør antall iterasjoner:
  - Setningene utføres så lenge betingelse er **True**
- **for**-løkke brukes for et bestemt antall iterasjoner
  - F.eks. for alle elementer i ei liste eller intervall

- Løkker kan nøstes inni hverandre
  - Se øving 3 for eksempler på dette
  - Og flere eksempler utover i semesteret