

# Python: Løkker

TDT4110 IT Grunnkurs  
Professor Guttorm Sindre

# Referansegruppe

- MTTK: [havardmellbye@gmail.com](mailto:havardmellbye@gmail.com)
  - MTMT: [Daniel.Vadseth@hotmail.com](mailto:Daniel.Vadseth@hotmail.com)
  - MTDESIGN [9valinn@gmail.com](mailto:9valinn@gmail.com)
  - MTKOM: [jonbs@stud.ntnu.no](mailto:jonbs@stud.ntnu.no)
  - MLREAL: [emmapet@stud.ntnu.no](mailto:emmapet@stud.ntnu.no)
  - BMUST?
- 
- På tide å ha et møte snart (neste uke?)
  - Begynn gjerne å tenke på relevant input

# Denne uka



Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•...sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=... +, -, *... >, ==, ...	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	<b>Kontrollstruktur</b> •standard sekvens •if-setning •løkker (while, for)	<b>Kontrollstruktur</b>  •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

# Læringsmål og pensum



- Mål
  - Forstå løkker i programmering
  - Ha kjennskap to ulike typer løkker (while-løkke, for-løkke)
    - Kunne velge type avhengig av behov
  - Praktiske ferdigheter i bruk av **while**-løkke
  - Praktiske ferdigheter i bruk av **for**-løkke
- Pensum
  - Starting out with Python:  
Chapter 4 Repetition Structures

# Løkker - gjenta kodelinjer



- HVORFOR: repetere handlinger
  - Uten at vi må skrive / kopiere kodelinjer mange ganger
    - Jfr. kopiering av formler, vanlig teknikk i regneark
- Repetisjoner i programmering: to teknikker
  - Løkker (i dag)
  - Rekursjon (mot slutten av semesteret)
- Løkker i Python: to typer
  - while-løkke
    - kan brukes på alle slags løkkeproblemer
  - for-løkke
    - ofte enklere for problemer hvor den passer

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

# Typisk bruk av ulike løkker

- **for**-løkke: kjent antall repetisjoner
  - fast antall, eller kjent (max.) 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/styringsystemer: Til en ønsket tilstand er nådd
    - Søking: Til ønskede data er funnet
    - Spill: Til noen har vunnet
    - ...

# EKS.: Kopiering vs. while vs. for

- Vil tegne et kvadrat på skjermen
- Pythons standardbibliotek mangler funksjoner for tegning
  - `print()` skriver bare tekst/tall til skjerm
- Må importere et tegnebibliotek, f.eks. turtle

```
File Edit Format Run Options Windows Help
```

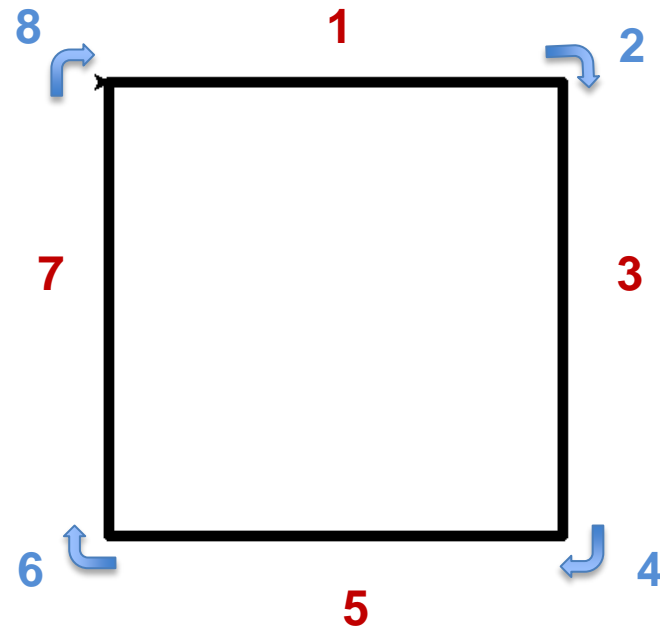
```
# Dette programmet tegner et kvadrat på skjermen
# Størrelse 300 x 300 piksler
import turtle          # importerer et bibliotek med tegnefunksjoner
turtle.pensize(7)     # setter større linjetykkelse

# TEGNING AV KVADRATET; GJENTAR TO SAMME KODELINJER 4 GANGER:
turtle.forward(300)   # første kant: rett strek mot høyre
turtle.right(90)      # første vinkel: vrir 90 grader med klokka
turtle.forward(300)   # andre kant: strek rett nedover
turtle.right(90)      # andre vinkel: vrir 90 grader med klokka
turtle.forward(300)   # tredje kant: strek rett mot venstre
turtle.right(90)      # tredje vinkel: vrir 90 grader med klokka
turtle.forward(300)   # fjerde kant: strek rett oppover
turtle.right(90)      # fjerde vinkel (tilbake til startretning)
```

KODE: kvadrat\_V0.py

# Forklaring av den repetitive delen

<code>turtle.forward(300)</code>	→ 1.	Tegner
<code>turtle.right(90)</code>	→ 2.	Vrir
<code>turtle.forward(300)</code>	→ 3.	Tegner
<code>turtle.right(90)</code>	→ 4.	Vrir
<code>turtle.forward(300)</code>	→ 5.	Tegner
<code>turtle.right(90)</code>	→ 6.	Vrir
<code>turtle.forward(300)</code>	→ 7.	Tegner
<code>turtle.right(90)</code>	→ 8.	Vrir



Funker, men...

**Hva er problemet med kopiering?**



# Samme med **while**

```
antall = 0
```

```
while antall < 4:
```

```
    turtle.forward(300)
```

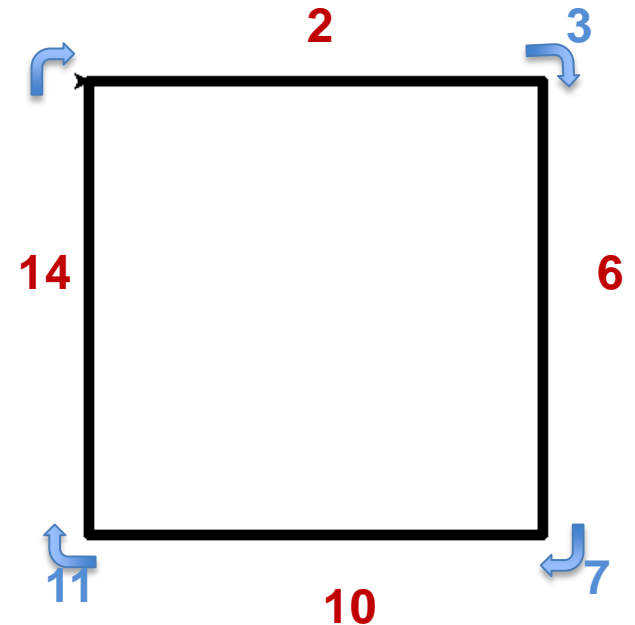
```
    turtle.right(90)
```

```
    antall = antall + 1
```

0	antall = 0
1	antall < 4 ? True
2	Tegn strek
3	Vri 90 grader
4	antall = 1
5	antall < 4 ? True
6	Tegn strek
7	Vri 90 grader
8	antall = 2
9	antall < 4 ? True
10	Tegn strek
11	Vri 90 grader
12	antall = 3
13	antall < 4 ? True
14	Tegn strek
15	Vri 90 grader
16	antall = 4
17	antall < 4 ? False

Fordeler vs.  
kopiering?

```
turtle.forward(300)
turtle.right(90)
turtle.forward(300)
turtle.right(90)
turtle.forward(300)
turtle.right(90)
turtle.forward(300)
turtle.right(90)
```



KODE: kvadrat\_while.py

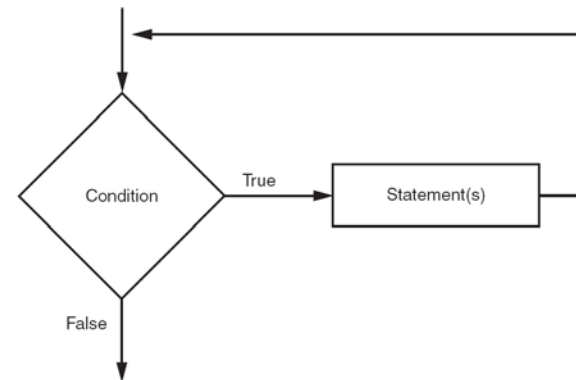
# Oppsett av while-løkke (kap 4.2)

**while** betingelse:

kodelinje1

kodelinje2

kodelinje3



- Minner om **if**:
  - Kun kode med innrykk hører til løkka
    - hvis betingelse er **True** blir linjer med inntrykk 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 (**if**: null eller én gang)
  - Hvis betingelsen **aldri** blir **False**: "evig løkke"
    - Bruk **Ctrl-C** for å avslutte programmet

# Samme med for

Lager tall-  
sekvensen  
0, 1, 2, 3

```
for antall in range(4):
```

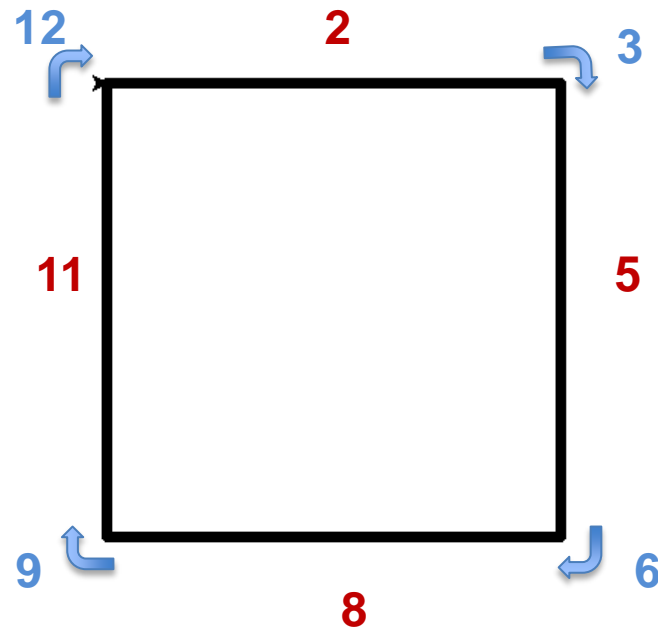
```
    turtle.forward(300)
```

```
    turtle.right(90)
```

## Fordeler vs. while?

```
antall = 0
while antall < 4:
    turtle.forward(300)
    turtle.right(90)
    antall = antall + 1
```

1	antall = 0 (første tall i sekvensen)
2	Tegn strek
3	Vri 90 grader
4	antall = 1
5	Tegn strek
6	Vri 90 grader
7	antall = 2
8	Tegn strek
9	Vri 90 grader
10	antall = 3
11	Tegn strek
12	Vri 90 grader
13	Hele sekvensen 0,1,2,3 er gjennomløpt, løkka er ferdig



KODE: kvadrat\_for.py

# for-løkker (kap. 4.3, 4.4)

- Gjenta ei kodeblokk et kjent / begrenset antall ganger
  - Gå gjennom en sekvens, element for element

- Generelt format:

```
for variabel in sekvens:  
    kodelinjer
```

- **sekvens** kan være
  - En sammensatt dataverdi (f.eks. liste, tuppel, mengde, ...)
    - 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**
  - En tekststreng
  - Et **range()**-objekt
- **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

KODE: kortstokk.py

# Fordeler med løkker vs. kopiering

- Kort kode selv med mange repetisjoner
- Fleksibilitet:
  - Antall repetisjoner kan **kontrolleres med variable**
  - Eksemplet: kort vei til at bruker kan velge hva som tegnes

Variabel for  
repetisjoner  
(ikke mulig m kopi)

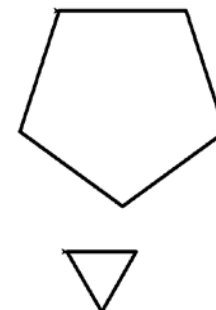
polygon\_for.py

```
for antall in range(4):  
    turtle.forward(300)  
    turtle.right(90)
```

Variable for  
kantlengde og  
vinkel (også mulig i  
kopi-eksemplet)

```
File Edit Format Run Options Windows Help  
# Dette programmet tegner regulære polygoner et  
import turtle # importerer et bibliotek m  
turtle.pensize(7) # setter større linjetykkelse  
  
print('Jeg kan tegne regulære polygoner')  
omkrets = int(input('Omkrets i piksler? '))  
kanter = int(input('Antall kanter? '))  
sidelengde = omkrets // kanter  
vinkel = 360 / kanter  
  
for antall in range(kanter):  
    turtle.forward(sidelengde)  
    turtle.right(vinkel)
```

```
>>> =====  
>>>  
Jeg kan tegne regulære polygoner  
Omkrets i piksler? 1200  
Antall kanter? 5  
>>> =====  
>>>  
Jeg kan tegne regulære polygoner  
Omkrets i piksler? 400  
Antall kanter? 3  
>>>
```



# Mer eksempler

```
>>> ===== RESTART =====
>>>
Jeg kan tegne "spiraler" av regulære polygoner
Omkrets i piksler? 1200
Antall kanter? 3
Reduksjon i lengde per strek? 20
>>> |
```

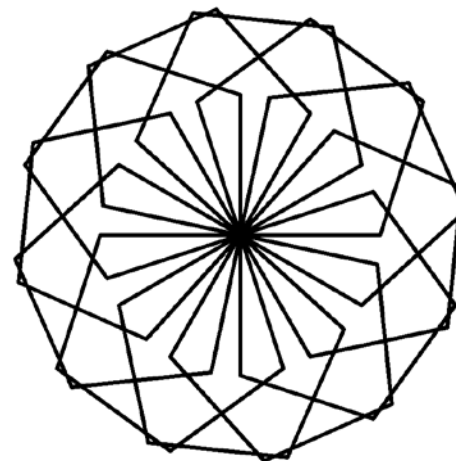
```
while sidelengde > 0:
    turtle.forward(sidelengde)
    turtle.left(vinkel)
    sidelengde = sidelengde - reduksjon
    # siste linje gjør strekene gradvis kortere
```

Dette er bare løkkedelen av programmet.  
Hele fins i fila [polygonspiral\\_while.py](#)



```
>>>
Jeg kan tegne mandalaer ved å rotere polygoner
Omkrets for polygonet? 1200
Antall kanter i polygonet? 5
Antall polygon i figuren? 12
```

```
# Ytre løkke gjør at vi får tegnet n polygoner
for n in range(ant_pol):
    # Indre løkke tegner ett polygon
    for antall in range(kanter):
        turtle.forward(sidelengde)
        turtle.left(vinkel)
    # Neste linje er i ytre løkke, endrer retning
    turtle.left(rotasjon)
```



Hele på fila  
[mandala\\_for.py](#)

# Oppgaver

## LETTERE:

Lag et program som skriver barnesangen "Fiskebollen lever i havet". Les inn ønsket antall vers fra brukeren og skriv hele teksten på skjermen.

F.eks. bruker velger 100:

```
'....Det var det 1. verset, nå er det bare 99 igjen'
```

```
... (div. vers...)
```

```
'...Det var det 100. verset, nå er det bare 0 igjen'
```

Prøv å få til en versjon med `for` og en med `while`.

Ufullstendig startkode på [fiskebollen\\_v0.py](#)

## MIDDELS:

(a) Lag et program som leser inn 10 flyttall fra tastatur (ett og ett). Deretter skal programmet skrive ut summen og produktet av tallene.

(b) i stedet for akkurat 10 flyttall, les inn et vilkårlig antall positive flyttall, avslutt hvis bruker gir et negativt tall. Fortell da hva som var det største tallet som ble skrevet inn.

Ufullstendig startkode på [tallrekke\\_v0.py](#)

## VANSKELIG:

Lag et program som leser inn et positivt heltall fra bruker via tastatur, og skriver ut på skjerm tallets faktorisering i primtall.

F.eks.,

```
'33 = 3*11'
```

```
'40 = 2*2*2*5'
```

Ufullstendig start: [faktorisering\\_v0.py](#)

# Bruk av range-objekter (kap 4.4)

- Et range-objekt husker en sekvens av tall
  - Lagrer **ikke** alle tallene
    - bare startverdi, sluttverdi og stegverdi (hvis ulik 1)
    - Dette sparer plass for lange tallrekker
  - Opprettes ved et kall til konstruktørmetoden `range()`
  - `range()` kan ta ett, to eller tre argumenter
    - Ett argument: tolkes som sluttverdi
      - Start default 0, steg default 1, siste element blir **sluttverdi - 1**
    - To argumenter: Tolkes som startverdi, sluttverdi
    - Tre argumenter: startverdi, sluttverdi, stegverdi
    - NB: alle må være heltall (int)

- Illustrasjon:

<code>range(8)</code>	<code># start: 0, slutt: 8, sekvensen blir 0, 1, 2, 3, 4, 5, 6, 7</code>
<code>range(2, 8)</code>	<code># sekvensen blir 2, 3, 4, 5, 6, 7</code>
<code>range(3, 20, 4)</code>	<code># sekvensen blir 3, 7, 11, 15, 19</code>
<code>range(21, 5, -3)</code>	<code># sekvensen blir 21, 18, 15, 12, 9, 6</code>



# Endring av verdier i variable

## Kompakte former (kap 4.5)



Lang-form	Kompakt-form	Hva gjøres
<code>x = x + 4</code>	<code>x += 4</code>	Øker verdien av x med 4
<code>x = x - 3</code>	<code>x -= 3</code>	Minsker verdien av x med 3
<code>x = x * 10</code>	<code>x *= 10</code>	Multipliserer verdien av x med 10
<code>x = x / 2</code>	<code>x /= 2</code>	Dividerer verdien av x med 2
<code>x = x % 4</code>	<code>x %= 4</code>	Resten av x delt på 4

Dette kan brukes også ellers i koden, ikke bare i løkker.  
Men er særlig relevant i løkker.

# Validering av input vha løkker (4.6)

- Brukere kan gi feil input
  - Med vilje
  - Tastefeil
  - Misforståelser
- Må sjekke input før programmet går videre
  - Evt. også tvinge brukeren til korrekt input
  - Dette kan man gjøre vha while-løkke:

```
alder = input("Hvor gammel er du? ")
while not alder.isdigit() or int(alder) < 0 or int(alder) > 150:
    print("Feil: Alder må være et heltall mellom 0 og 150!")
    alder = input("Hvor gammel er du? ")
print("Takk for den du!! ")
alder = int(alder)
```

**kode: alder.py**

# Nøstede løkker (kap 4.7)

- Løkke inni annen løkke kalles nøstede løkker.
- Nyttig i mange situasjoner
  - Hierarkiske strukturer
  - Tabeller, matriser
- Tid er et godt eksempel
  - teller først 60 sekunder,
  - øker minutt med 1 osv...
- Utskrift av tid som nøstede løkker:

```
for t in range(24):  
    for m in range(60):  
        for s in range(60):  
            print(t,":",m,":",s)
```

**kode: tid.py**

# Oppsummering



- while-løkke: en betingelse avgjør antall iterasjoner:
  - while(betingelse):  
setning(er)  
Setningene utføres så lenge betingelse er True
- for-løkke brukes for et bestemt antall iterasjoner
  - F.eks. gjøre noe for alle elementer i ei liste eller intervall  
for x in [1, 2, 3, 4]:  
for y in ['test', 3.14, True, 9]:  
for i in range(1, 5):  
for z in range(1, 5, 2):
- Nøstede løkker er løkker inne i andre løkker:  
for x in [1, 3, 5]:  
    for y in range [5, 7, 12]:  
        ...