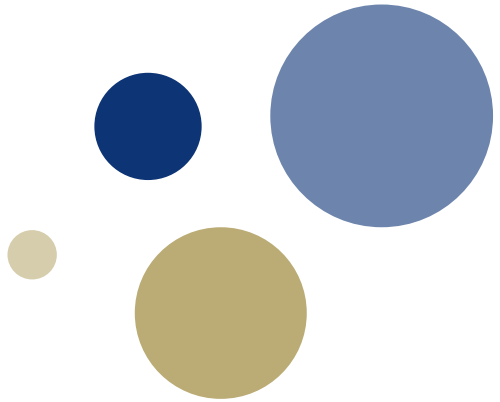
NTNU

Kunnskap for en bedre verden



Python: Løkker

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

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
Bestemme hvilke programsetninger som utføres, og hvor mange ganger • Valg • Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •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
  - Kjenne to ulike typer løkker (while, for)
    - Velge løkke avhengig av behov
    - Forstå hvordan de virker
    - Kunne programmere løkker som fungerer
      - Enkle løkker …. kompliserte nøstede løkker

- Pensum
  - Starting out with Python:

  Chapter 4 Repetition Structures

Løkker - gjenta kodelinjer

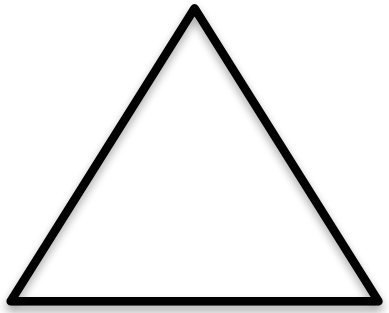
- HVORFOR: repetere handlinger
  - Men hvorfor ikke bare kopiere kodelinjer i stedet? (jfr. Excel)

File Edit Format Run Options Window Help

```
# importerer tegnefunksjoner
from turtle import *

# tykkere strek
pensize(7)

# tegner en trekant
forward(200)
left(120)
forward(200)
left(120)
forward(200)
left(120)
```



KODE: trekant_3_alternativ.py

Løkker - gjenta kodelinjer

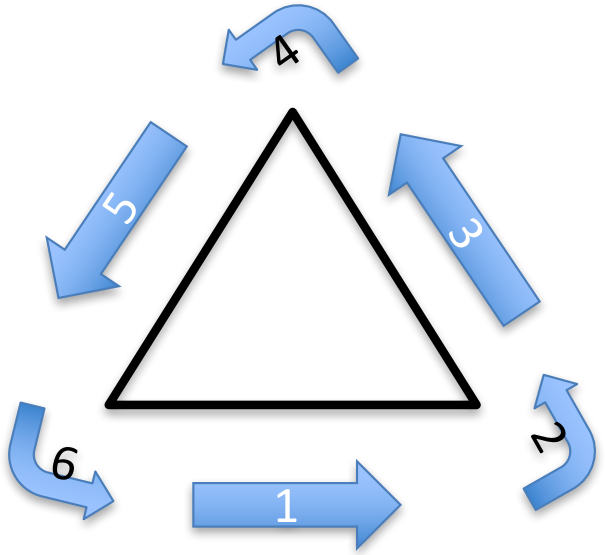
- HVORFOR: repetere handlinger
  - Men hvorfor ikke bare kopiere kodelinjer i stedet? (jfr. Excel)

File Edit Format Run Options Window Help

```
# importerer tegnefunksjoner
from turtle import *

# tykkere strek
pensize(7)

# tegner en trekant
forward(200)
left(120)
forward(200)
left(120)
forward(200)
left(120)
```



Løkker - gjenta kodelinjer

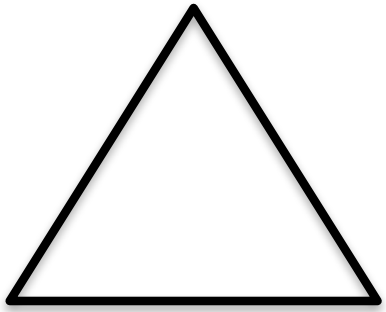
- HVORFOR: repetere handlinger
  - Men hvorfor ikke bare kopiere kodelinjer i stedet? (jfr. Excel)

File Edit Format Run Options Window Help

```
# importerer tegnefunksjoner
from turtle import *

# tykkere strek
pensize(7)

# tegner en trekant
forward(200)
left(120)
forward(200)
left(120)
forward(200)
left(120)
```



```
# samme trekant med while
kant = 0
while kant < 3:
    kant = kant + 1
    forward(200)
    left(120)
```

```
# samme trekant med for
for kant in range(3):
    forward(200)
    left(120)
```

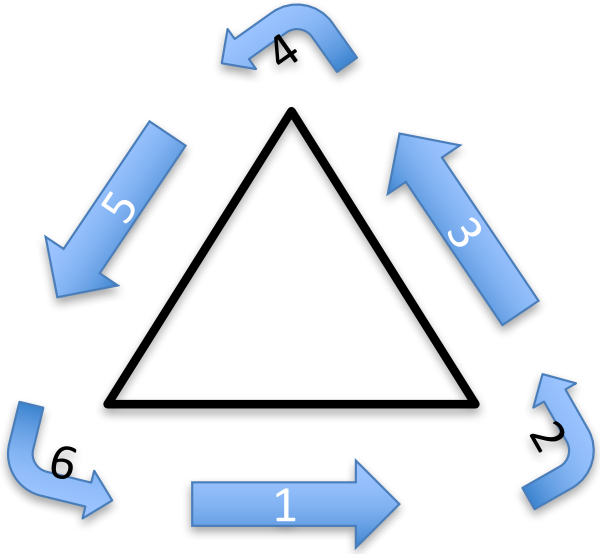
Typisk bruk av ulike løkker

- 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/styringssystemer: Til en ønsket tilstand er nådd
    - Søking: Til ønskede data er funnet
    - Spill: Til noen har vunnet
    - …

Trekant med while

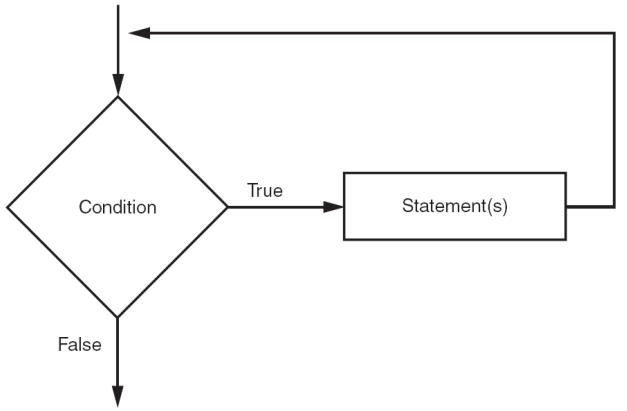
```
antall = 0
while antall < 3:
    forward(200)
    left(120)
    antall = antall + 1
```

0	antall = 0
1	antall < 3 ? True
2	Tegn strek
3	Vri 120 grader
4	antall = 0 + 1 (dvs. 1)
5	antall < 3 ? True
6	Tegn strek
7	Vri 120 grader
8	antall = 1 + 1 (dvs. 2)
9	antall < 3 ? True
10	Tegn strek
11	Vri 120 grader
12	antall = 2 + 1 (dvs. 3)
13	antall < 3 ? False



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 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”
    - Bruk **Ctrl-C** for å avslutte programmet

Trekant med for

Lager tall-sekvensen 0, 1, 2

```
for antall in range(3):
    forward(200)
    left(120)
```

1	antall = 0 (første tall i sekvensen)
2	Tegn strek
3	Vri 120 grader
4	antall = 1
5	Tegn strek
6	Vri 120 grader
7	antall = 2
8	Tegn strek
9	Vri 120 grader
10	Hele sekvensen 0,1,2 er gjennomløpt, løkka er ferdig

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 uten å utføre den

KODE: kortstokk.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
    - sparer plass for lange tallrekker
  - Opprettes ved 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:

```
range(8)        # start: 0, slutt: 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
```

Oppgaver

LETTERE:
La brukeren gi ønsket antall vers, skriv hele teksten til «Fiskebollen lever i havet» på skjermen. Eks. kjøring:

```
Antall vers? 3

Fiskebollen lever i havet.
Havet er fiskebollens venn.
Det var det 1. verset.
Nå er det bare 2 igjen.

Fiskebollen lever i havet.
Havet er fiskebollens venn.
Det var det 2. verset.
Nå er det bare 1 igjen.

Fiskebollen lever i havet.
Havet er fiskebollens venn.
Det var det 3. verset.
Nå er det bare 0 igjen.
```

Lag en versjon med for og en med while.

Ufullstendig startkode på fiskebollen_v0.py

MIDDELS:
(a) Lag et program som lar bruker gi inn 10 flyttall fra tastatur (ett og ett). Så skal programmet skrive ut summen og produktet av tallene.

(b) I stedet for akkurat 10 flyttall, motta 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.,

```
Jeg kan faktorisere.
Gi meg et heltall? 40
40 er 2*2*2*5
```

Ufullstendig start: faktorisering_v0.py

Løsning på oppgaver

- ”Lettere”: skal skrive (hvis bruker sa 100):

```
...det 1. verset...99 igjen.
...det 2. verset...98 igjen.
...
...det 100. verset...0 igjen.
```

- Hvordan tenke her?
  - Gjenta utskrift N ganger: LØKKE
  - Antall gjentak kjent: for-løkke enklest
    - antall vers fra input() → variabel ant_vers
  - Hva med løkke for vers in range(ant_vers): ?
    - Gir tallserien (i) 0, 1, 2, …, ant_vers – 1
    - Trenger (ii) 1, 2, 3, …, ant_vers (nr på verset)
    - og (iii) ant_vers – 1, ant_vers – 2, …, 1, 0 (antall igjen)
    - Kan vi oppnå (ii) og (iii) fra (i) samt andre data vi har?
      - Ja! (vers + 1 gir nr, ant_vers – vers – 1 gir gjenstående)
      - Eller skrive for vers in range(1, ant_vers + 1), da blir de to andre
        - » vers og ant_vers – vers

Løsning: fiskebolle.py

Løsning på oppgaver (2)

- ”Middels”:
  - A) lese 10 tall fra tastatur, fortelle summen og produktet
  - Hvordan tenke her?
    - Gjenta lesing av tall 10 ganger: Løsning er løkke
    - Kjent antall gjentak: **for-løkke** passer, for i in range(10): vil kjøre 10 runder
    - Hvilke variable trenger vi? En for tallet som leses inn, en for summen, en for produktet
      - Trenger IKKE lage 10 forskjellige variable eller liste for å huske alle innleste tall etterpå, dette er ikke bedt om
    - Typisk oppsamlingsproblem (gradvis samle et resultat ved utførelse av løkke).
    - Felles for slike problem
      - Oppsamlingsvariable må opprettes FØR løkka
        - » For en sum er riktig startverdi 0, for produkt 1
      - …oppdateres i hver runde av løkka
        - » Addere inn / multiplisere inn det neste tallet
      - …og leveres (her skrives ut) etter at løkka er ferdig
  - B) lese vilkårlig antall tall fra tastatur, inntil negativt tall blir gitt. Fortelle maksimum
  - Hvordan tenke her?
    - Antall repetisjoner er vilkårlig: **while-løkke**
    - Oppsamling: finne maksimum, for hver runde sjekke om nytt tall er større
      - Trenger en variabel for det hittil største tallet
      - if-setning hvor denne variabelen endres hvis nytt tall var større
    - beste startverdi det første tallet, variabelen må opprettes før løkka

Løsning: tallrekke.py

Løsning på oppgaver (3)

- ”Vanskelig”:
  - Skrive ut primtallsfaktoriseringen av et innlest heltall x, f.eks
    - 19 er 19
    - 6 er 2 * 3
    - 40 er 2 * 2 * 2 * 5
  - Hvordan tenke her?
    - Prøve ett og ett tall, finne ut om det er en faktor (x % divisor == 0)
    - Hvis faktor funnet, dele tallet på faktoren (x = x // divisor)
      - Så vi vet resten som fortsatt gjenstår å faktorisere
    - Hvordan vite at vi skal slutte? Når x er blitt 1
    - Samme tall kan være faktor flere ganger. Må prøve om igjen til det ikke lenger er faktor
    - Dette blir en dobbel løkkekonstruksjon:
      - Ytre løkke: prøve økende divisorer 2, 3, 4, 5, 6, 7, … så lenge x > 1
      - Indre løkke: prøve samme divisor gjentatte ganger, så lenge x % divisor == 0
    - Med tanke på genereringen av tekststrengen som skal bli svaret, er dette et typisk oppsamlingsproblem
      - Gi den en startverdi før løkka (f.eks ’40 = ’)
      - I løkka: legge til hver faktor vi finner bakerst i strengen, samt ’*’ unntatt for aller siste faktor
      - Printe strengen som løsning etter at løkka er slutt

Løsning: faktorisering.py

Endring av verdier i variable
Kompakte former (kap 4.5)

Lang-form	Kompakt-form	Hva gjøres
`x = x + 4`	`x += 4`	Øker verdien av x med 4
`x = x - 3`	`x -= 3`	Minsker verdien av x med 3
`x = x * 10`	`x *= 10`	Multipliserer verdien av x med 10
`x = x / 2`	`x /= 2`	Dividerer verdien av x med 2
`x = x % 4`	`x %= 4`	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, 9, 2):
```

- Nøstede løkker er løkker inni andre løkker:

```
for x in [1, 3, 5]:
    for y in range [5, 7, 12]:
        ...
```