KAPITTEL 5

Løkker

5.1 Enkel for-løkke (5_for.py)

```
1 for tall in range(3):
2     print(tall)
```

Vi kan bruke en `for` -løkke for å gjenta en eller flere kodelinjer. Du skal nå utforske hvordan en slik løkke fungerer.

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_for.py» og kjør programmet.

Nedenfor ser du fire kodebiter som inneholder feil.

```
for tall range(3):
    print(tall)
```

```
for tall in range 3:
    print(tall)
```

```
for tall in range(3)
    print(tall)
```

```
for tall in range(3):
print(tall)
```

c) Bestem feilen i hvert av de fire tilfellene ovenfor.

d) Endre `range(3)` til `range(5)` . Kjør programmet.

e) Endre `range(5)` til `range(8)` . Gjett på hvilket tall som blir skrevet ut til sist.

f) Endre `range(8)` til `range(2, 8)` .

g) Endre `range` til `range(-10, 0)` .

h) Skriv ut alle hele tall fra og med -5 til og med 10.

i) Skriv ut alle hele tall fra og med -100 til og med 100.

5.2 Tallmønstre med addisjon, metode 1 (5_addsub1.py)

```
1 for tall in range(3, 8):
2     print(tall, end=" ")
```

Vi kan bruke for-løkker til å lage tallmønstre der differansen mellom to naboledd er konstant. For eksempel $2, 5, 8, 11, \ldots$.

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_addsub1.py» og kjør programmet.

c) Endre `range(3, 8)` til `range(3, 8, 2)`. Kjør programmet på nytt.

Gjett alltid på resultatet etter hver endring, før du kjører programmet på nytt.

d) Endre `range` til `range(-4, 8, 2)`.

e) Endre `range` til `range(-4, 8, 3)`.

f) Endre `range` slik at resultatet blir

```
-7 -4 -1 2 5
```

g) Endre `range` slik at resultatet blir

```
-7 -4 -1 2 5 8
```

h) Endre `range` slik at resultatet blir

```
19 16 13 10 7 4 1
```

i) Fullfør programmet nedenfor slik at kjøringen av programmet gir utskriften til høyre.

```
for tall in range(...):
    print(tall, end=" ")
print()
print("Motsatt vei"))
for tall in range(...):
    print(tall, end=" ")
```

```
1 5 9 13 17 21 25 29 33 37 41
Motsatt vei:
41 37 33 29 25 21 17 13 9 5 1
```

5.3 Tallmønstre med addisjon, metode 2 (5_addsub2.py)

```
1 tall = 1
2 for i in range(4):
3     print(tall)
4     tall = tall + 2
```

```
Gi tall verdien 1
Gjenta 4 ganger:
    Skriv ut tall
    Øk tall med 2
```

Ovenfor ser du kode og algoritme ved siden av hverandre. Algoritmen skrives på norsk, mens koden skrives på Pythonsk.

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_addsub2.py» og kjør programmet.

c) Endre `tall = 1` til `tall = 5`. Gjett på resultatet, før du kjører programmet på nytt.

d) Endre `range(4)` til `range(6)`.

e) Endre `tall = tall + 2` til `tall = tall + 5`.

f) Endre `tall = tall + 5` til `tall = tall - 3`.

g) Endre koden slik at resultatet blir

```
15 12 9 6 3 0
```

Bruk `print(tall, end=" ")`.

h) Endre koden slik at resultatet blir

```
15 12 9 6 3 0 -3 -6 -9
```

i) Endre koden slik at resultatet blir

```
74 78 82 86 90 94 98 102
```

Et tallmønster er gitt ved $-12, -5, 2, 9, \ldots$.

j) Endre koden slik at du skriver ut de 50 første leddene i dette tallmønsteret. Resultatet skal bli:

```
-12 -5 2 9 16 23 30 37 44 51 58 65 72 79 86 93 100 107 114 121
128 135 142 149 156 163 170 177 184 191 198 205 212 219 226 233
240 247 254 261 268 275 282 289 296 303 310 317 324 331
```

5.4 Gange- og delemønstre (5_multdiv.py)

```
1 tall = 5
2 for i in range(4):
3     tall = tall*2
4 print(tall)
```

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_multdiv.py» og kjør programmet.

c) Endre plasseringen av `print(tall)` som vist nedenfor. Gjett på resultatet, før du kjører programmet på nytt.

```
for i in range(4):
    tall = tall*2
    print(tall)
```

d) Endre `tall = tall*2` til `tall = tall*3`.

e) Endre `tall = tall*3` til `tall = tall/2`.

f) Endre koden slik at programmet skriver ut

```
2.5
1.25
0.625
0.3125
0.15625
0.078125
```

g) Endre koden slik at programmet skriver ut

```
10.0
5.0
2.5
1.25
0.625
```

h) Endre koden slik at programmet skriver ut

```
1 3 9 27 81 243 729 2187 6561 19683 59049 177147
```

Bruk blant annet `print(tall, end=" ")`.

5.5 Summere med løkker (5_sum.py)

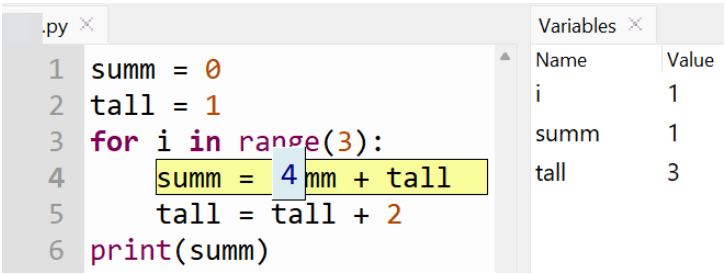
```
1 summ = 0
2 tall = 1
3 for i in range(3):
4     summ = summ + tall
5     tall = tall + 2
6 print(summ)
```

For å holde styr på hvordan verdiene til variablene utvikler seg, velger du View » Variables i menyen til Thonny.

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_sum.py» og kjør programmet.

c) Kjør feilsøking. Bruk «step over» på de to første linjene. Når du kommer til `for`-løkka, trykker du «step into» gjentatte ganger for å gå gjennom resten av programkjøringen.



Figur 5.1: Feilsøking for-løkke.

d) Kjør feilsøking på tilsvarende måte en gang til. Denne gangen følg med på hvordan verdiene til variablene utvikler, og forsøk å forutsi hva som vil skje på hver kodelinje. Se figur 5.1.

e) Endre koden slik at programmet regner ut summen $1+3+5+7+9$. Sjekk at svaret blir 25.

f) Endre koden slik at programmet regner ut følgende sum:

$$5+8+11+14+\cdots+59+62$$

Sjekk at svaret blir 670.

5.6 While-løkke: sparing (5_while.py)

```
saldo = 0
while saldo < 100_000:
    saldo = saldo + 3000
print(saldo)
```

Susanna sparer et fast beløp hver måned. Vi antar først at beløpet er 3000 kr, og at hun ønsker å spare til 100 000 kr. Vi ser bort fra renter.
Når vi er usikre på antall gjentakelser, men kriteriet for å stoppe løkka er kjent, er det best å bruke en `while`-løkke.

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_while.py» og kjør programmet.

Dersom resultatet av programmet blir `0`, er det sannsynligvis fordi du har skrevet `saldo > 100_000` i stedet for `saldo < 100_000`.

c) Legg til variabelen `sparebelop` slik:

```
sparebelop = 3000
saldo = 0
while saldo < 100_000:
    saldo = saldo + sparebelop
print(saldo)
```

Det neste vi må gjøre, er å telle opp antall måneder hun sparer.

d) Legg til følgende tre kodelinjer på riktig plass, der bare én av dem skal skrives inni løkka:

`mnd = 0` `print("Antall måneder:", mnd)` `mnd += 1`

Kodelinja `mnd += 1` er identisk med `mnd = mnd + 1`.

For å teste at vi får korrekt antall måneder, kan vi velge for eksempel 7000 kr.

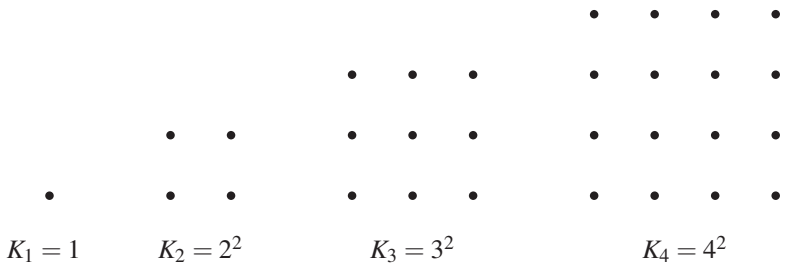
e) Endre stoppkriteriet til `saldo < 7000`. Dette bør gi 3 måneder.

Susanna bestemmer seg for å spare 4600 kr hver måned. Hun vil spare til 250 000 kr.

f) Endre programmet slik at vi finner ut hvor lang tid Susanna må spare.

Ekstra: Forsøk gjerne å svare i år og måneder. Bruk blant annet `//`.

5.7 Figurtall (5_figtall.py)



Figur 5.2: Kvadrattallene med prikker.

Figur 5.2 viser de fire første kvadrattallene. Vi ser for eksempel at det tredje kvadrattallet K_3 har side lik 3, og har derfor $3^2 = 9$ prikker.

a) Les koden nedenfor, og gjett på resultatet.

```
for fignr in range(1, 5):
    K = fignr**2
    print(f"K_{fignr} = {K}")
```

b) Skriv av koden, lagre som «5_figtall.py» og kjør programmet.

c) Endre på `range(1, 5)` slik at vi ser antall prikker i de 10 første kvadrattallene.

d) Legg til føldende kodelinjer på riktig plass for å summere de 10 første kvadrattallene:

`prikker_totalt = 0` `prikker_totalt += K`

`print("Prikker totalt:", prikker_totalt)`

Pia har lyst til å tegne til sammen 3000 prikker. Hun tegner først K_1, så K_2 og så videre.

e) Bruk programmet til å utforske hvor mange figurer Pia kan tegne. Vi bestemmer figurnummeret på den siste K-en Pia tegner.

Oppgave e) er en oppgave der man typisk bruker en `while` -løkke, fordi stoppkriteriet (maks 3000) er kjent mens antall gjentakelser er ukjent. Forsøk gjerne å løse den med `while` -løkke også, etter at du har funnet svaret på e).

5.8 While True: prøv igjen (5_gjett.py)

```
1 import random
2
3 tall = random.randint(0, 100)
4 gjett = int(input("Gjett på tallet (0-100):"))
5 if gjett > tall:
6     print("Du gjettet for høyt")
```

Vi skal lage et program der brukeren kan gjette på et tilfeldig generert tall fra 0 til 100. Ved hjelp av en løkke kan brukeren også få flere forsøk på å gjette.

a) Les koden ovenfor, og gjett på resultatet.

b) Skriv av koden, lagre som «5_gjett.py» og kjør programmet.

c) Legg til kodelinja `print(tall)` rett under `gjett = ...`. Slik vil vi kunne se det korrekte tallet mens vi holder på med å teste programmet.

d) Utvid `if`-setningen som vist nedenfor, og kjør programmet.

```
if gjett > tall:
    print("Du gjettet for høyt")
elif gjett < tall:
    print("Du gjettet for lavt")
```

e) Avslutt `if`-setningen med `else:`, og legg til en `print`-kommando med teksten `Du gjettet riktig!`.

Kommandoen `while True:` lar oss gjenta kode et ubegrenset antall ganger.

f) Utvid koden som vist nedenfor, og kjør deretter programmet.

```
tall = randint(0, 100)
while True:
    gjett = int(input("Gjett på tallet (0-100):"))
    if gjett > tall:
        print("Du gjettet for høyt")
    # Og så videre
```

Trykk 

dersom du ønsker å stoppe kjøringen av programmet.

Figur 5.3: Illustrasjon: Hemmelig tall.

Vi skal nå utvide programmet slik at vi teller opp antall forsøk på å gjette riktig.

g) Utvid koden som vist nedenfor slik at antall forsøk øker med 1 hver gang brukeren gjetter.

```
tall = randint(0, 100)
antall_gjett = 0
while True:
    gjett = int(input("Gjett på tallet (0-100):"))
    antall_gjett += 1
    if gjett > tall:
```

h) Legg til kode slik at når brukeren svarer riktig, skal han få vite antall forsøk han brukte. Kjøringen av programmet kan se slik ut:

```
Gjett på tallet (0-100): 60
Du gjettet for høyt
Gjett på tallet (0-100): 30
Du gjettet for lavt
Gjett på tallet (0-100): 55
Riktig
Du brukte 3 forsøk
```

i) Legg til kodelinja `break` på stedet i koden der brukeren har svart riktig. Slik stopper vi `while`-løkka.

j) Endre på programmet slik at brukeren får maksimalt 5 forsøk. Dersom forsøkene er brukt opp, skal det korrekte tallet avsløres og spillet avsluttes. Du kan få bruk for kodelinjene nedenfor.

```
antall_forsok -= 1
```

```
antall_forsok = 5
```

```
if antall_forsok == 0:
```

```
print("Game over")
```

5.9 Oppsummering kapittel 5

Vi lærte ditt og datt

5.10 Oppgaver

Oppgave 5.1A

(5_opg1A.py)

`for tall in range(4):` `print(tall)`

Sett sammen kodelinjene for å lage et program som skriver ut tallene $0, 1, 2$ og 3.

Oppgave 5.2A

(5_opg2A.py)

```
For i in renge(5, 9]
    pint(i)
```

Rett alle feilene i koden slik at programmet fungerer.

Oppgave 5.3A

(5_opg3A.py)

```
1 for tall in range(...):
2     print(..., end=" ")
```

Fullfør koden slik at resultatet blir:

```
0 1 2 3 4 5 6 7 8 9 10
```

Oppgave 5.4A

(5_opg4A.py)

```
1 for i in range(...):
2     print(..., end=" ")
```

Fullfør koden slik at resultatet blir:

```
12 7 2 -3 - 8 -13 -18
```

Oppgave 5.5A

(5_opg5A.py)

`tall = 44` `tall = tall - 7` `print(tall)`

`while tall > 0:` `print(tall, end=", ")`

Sett sammen kodelinjene for å lage et program som gir resultatet nedenfor.

```
44, 37, 30, 23, 16, 9, 2, -5
```

Oppgave 5.6A

(5_opg6A.py)

```
tall = ...
while ... < 525_000:
    print(...)
    tall = tall * ...
```

Fullfør koden slik at resultatet blir:

```
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192
16384 32768 65536 131072 262144 524288
```

Oppgave 5.7A

(5_opg7A.py)

Unni inngår en avtale med sin far om å få ukelønn etter følgende regler:

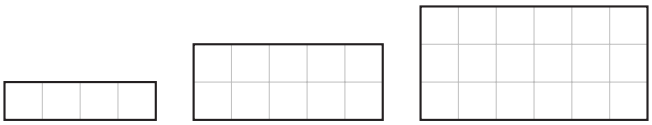
- 200 kr den første uka.
- Ukalønna øker deretter med 3 kr per uke.

Fullfør programmet nedenfor. Skriv ut total ukelønn i løpet av det første året.

```
total_lonn = 0
ukelonn = 200
for i in range(52):  # 52 uker i ett år
    total_lonn += ...
    ukelonn += ...
```

Oppgave 5.8A

(5_opg8A.py)



$R_1 = 4$ $R_2 = 10$ $R_3 = 18$

Figur 5.4: Figurtall rektangler.

Et tallmønster $\{R_n\}$ er gitt ved figur 5.4. Bruk kodelinjene nedenfor til å lage et program som skriver ut antall kvadrater i figuren for de første 10 figurene.

```
bredde = hoyde + 3
```

```
hoyde = fignr
```

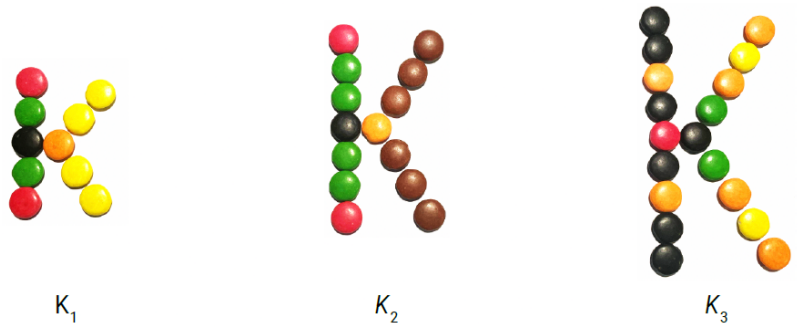
```
print(f"R_{fignr} = {figur}")
```

```
figur = bredde * hoyde
```

```
for fignr in range(1, 11):
```

Oppgave 5.9A

(Eksamen 1P, vår 2023) (5_opg9A.py)



Figur 5.5: Non Stop figurtall.

Kari har brukt Non Stop og laget tre K-er. Se ovenfor. Tenk deg at hun skal fortsette å lage K-er etter samme mønster. Ved å telle Non Stop på figur 5.5, ser vi at

$$K_1 = 10 \quad K_2 = 14 \quad K_3 = 18$$

Kari ønsker å lage et program som finner antall Non Stop hun trenger for å lage hver av de 20 første K-ene. Hun ønsker også å vite hvor mange Non Stop hun trenger til sammen for å lage alle disse 20 K-ene.

Lag et program som Kari kan bruke. Du kan for eksempel begynne som vist nedenfor, men legge inn formler i stedet for tallet én i kodelinje 12 og 13 slik at den riktige oversikten skrives ut.

```
1  # Startverdier
2  nonstop_figur = 10
3  nonstop_totalt = 10
4
5  # Overskrifter
6  print("Figurnummer\t\tNon Stop i figur\tNon Stop totalt")
7
8  for figurnummer in range(1, 21):
9      # Skriver ut kolonner ved å bruke tabulatorer sep = "\t\t\t"
10     print(figurnummer, nonstop_figur, nonstop_totalt, sep="\t\t\t")
11
12     nonstop_figur = 1
13     nonstop_totalt = 1
```

Antall Non Stop øker med 4 fra en figur til den neste. Dermed må `nonstop_figur` *på kodelinje 12 øke med 4.*

Oppgave 5.10B

(5_opg10B.py)

Lag et program der du bruker en løkke til å regne ut

$$44+37+30+23+16+9+2+(-5)$$

Oppgave 5.11B

(5_opg11B.py)

Lag et program som regner ut summen av de 30 første leddene i summen

$$100+50+25+\cdots$$

Oppgave 5.12B

(5_opg12B.py)

En dyrepopulasjon består nå av 50 dyr. Populasjonen øker med 18 % hvert år. Skriv et program som bestemmer antall år det tar før populasjonen overstiger 300 dyr. Bruk blant annet `while dyr <= 300:` til å løse oppgaven.

Oppgave 5.13B

(5_opg13B.py)

Måned	Saldo
1	300
2	$300+300\cdot 1{,}005=601{,}50$
3	$300+601{,}5\cdot 1{,}005=904{,}51$

Tabell 5.1: Utregning månedlig sparing.

Anta at du setter inn 300 kr på en sparekonto hver måned. Månedlig rente er 0,5 %, som gir vekstfaktor 1,005. Lag et program som skriver ut saldoen på kontoen hver måned de første 20 månedene. Tabell 5.1 viser hvordan du regner.

Oppgave 5.14B 1T

(5_opg14B.py)

Formelen for det n-te tallet i en tallfølge er $a_n = n^2 - 3\cdot n + 2$. Lag et program som regner ut summen $a_1 + a_2 + a_3 + \cdots + a_{12}$

Oppgave 5.15B 1T

(5_opg15B.py)

Endre programmet i oppgave 5.14 slik at du undersøker hvor mange ledd du må ta med for at summen for første gang overstiger 10 millioner.

Oppgave 5.16B 1T

(5_opg16B.py)

```
tall = int(input(...))
primtall = True
for deletall in range(2, tall):
    # Hvis tall kan deles på deletall
        # Sett primtall til False
        # Avslutt løkka
if primtall:
    # Skriv ut tallet er et primtall
else:
    # Skriv ut tallet ikke er et primtall
```

Lag et program der brukeren kan skrive inn et positivt heltall større enn 2, og få vite om tallet er et primtall eller ikke. Bruk den halvferdige koden og skriv den ferdig.

Delelighet kontrolleres ved hjelp av rest-operatoren `%` *. For eksempel gir* `4 % 2` *0, mens* `5 % 2` *gir 1. Resten er alltid 0 dersom divisjonen går opp.*

Oppgave 5.17B

(5_opg17B.py)

Lag et program som genererer et tilfeldig heltall fra 0 til 121. Brukeren skal gjette på kvadratroten til tallet, med nøyaktighet på 1 desimal. Programmet skal fortelle brukeren om gjetningen er for lav eller for høy, og fortsette å be om gjetninger til brukeren gjetter riktig.

For å lage fasit avrundet til én desimal, kan du skrive `fasit = round(math.sqrt(tall), 1)` .

Oppgave 5.18B

(5_opg18B.py)

```
for i in range(...):
    ...
    for j in range(1, i):
        ...
```

Bruk strukturen i koden ovenfor og lag et program som skriver ut følgende mønster:

```
1*2 = 2
1*2*3 = 6
1*2*3*4 = 24
...
1*2*3*4*5*6*7*8*9*10 = 3628800
```

Oppgave 5.19S

(5_opg19S.py)

Figur 5.6: Illustrasjon: Collatz-problemet.

En matematiker utforsker et spesielt tallmønster. Hun starter med et vilkårlig positivt heltall. Hvis tallet er oddetall, multipliserer hun det med 3 og legger til 1. Hvis tallet er partall, deler hun det på 2. Hun gjentar prosessen med det nye tallet.

For eksempel, hvis hun starter med tallet 6, vil sekvensen bli: 6, 3, 10, 5, 16, 8, 4, 2, 1. Deretter vil sekvensen gjenta seg med 4, 2, 1, ...

Matematikeren merker at alle tallene hun har prøvd, ender opp i sekvensen 4, 2, 1. Hun lurer på hvor mange steg det tar for ulike starttall å nå 1 for første gang.[1]

Lag et program som skriver ut denne sekvensen for hvert av starttallene $3, 4, 5, \ldots, 20$. Det skal også skrives ut antall steg det tok for å nå 1 for første gang. Begynnelsen av utskriften skal se slik ut:

```
3, 10, 5, 16, 8, 4, 2, 1
Starttall: 3, steg: 7
4, 2, 1
Starttall: 4, steg: 2
5, 16, 8, 4, 2, 1
Starttall: 5, steg: 5
```

[1] Dette er et kjent problem i tallteori kalt Collatz-problemet. Foreløpig er det faktisk ikke kjent om alle positive heltall ender opp i sekvensen 4, 2, 1.

DEL II

Programmering i matematikk S1

DEL III

Programmering i matematikk S2