

KAARE ERLEND JØRGENSEN  
STEIN ALEXANDER DAHL

# **KAARES KOKEBOK**

## **PROGRAMMERING I MATEMATIKK**

ET KOMPLETT OPPLÆRINGSLØP PÅ VGS

Jørgensen & Dahl Forlag AS

© 1. utgave Jørgensen & Dahl Forlag AS 2023  
1. opplag 2023

ISBN 978-82-692219-9-2

Materialet i denne publikasjonen er vernet etter åndsverkloven. Uten uttrykkelig samtykke fra rettighetshaverne er eksemplarframstilling, som utskrift og annen kopiering, bare tillatt når det er hjemlet i lov eller avtale med Kopinor ([www.kopinor.no](http://www.kopinor.no)).

Utnyttelse i strid med lov eller avtale kan medføre erstatnings- og straffeansvar.

Boka har egen nettside: [www.jdforlag.no](http://www.jdforlag.no)

Henvendelser om denne utgivelsen kan rettes til: [kontakt@jdforlag.no](mailto:kontakt@jdforlag.no)

Omslag og omslagsillustrasjon: Kaare E. Jørgensen & Stein A. Dahl  
Sats, figurer og formgivning: Kaare E. Jørgensen & Stein A. Dahl  
Boka er satt med: Computer Modern 10/12, pdf<sub>l</sub>atex/TikZ/tcolorbox

ALL HAPPY PEOPLE ARE GRATEFUL.  
UNGRATEFUL PEOPLE CANNOT BE HAPPY.  
WE TEND TO THINK THAT BEING UNHAPPY LEADS PEOPLE  
TO COMPLAIN, BUT IT'S TRUER TO SAY THAT  
COMPLAINING LEADS TO PEOPLE BECOMING UNHAPPY.

*D. Prager*



# Innhold

## Innledning

xiii

<b>I</b>	<b>Programmering på Vg1</b>	<b>1</b>
<b>1</b>	<b>Skrive til skjerm</b>	<b>3</b>
1.1	Nedlasting av Thonny . . . . .	3
1.2	Installasjon av Thonny . . . . .	3
1.3	Skriv ut «Hallo, Norge» . . . . .	4
1.4	Skrive ut et dikt . . . . .	5
1.5	Regne med Python . . . . .	6
1.6	Potenser . . . . .	7
1.7	Kvadratrøtter . . . . .	8
1.8	Oppsummering kapittel 1 . . . . .	9
1.9	Oppgaver . . . . .	10
<b>2</b>	<b>Variabler</b>	<b>13</b>
2.1	Lagre tekst i en variabel . . . . .	13
2.2	Lagre tall i variabler . . . . .	14
2.3	Bruke variabler i utregning . . . . .	15
2.4	Feilsøking i Thonny . . . . .	16
2.5	Endre en variabel . . . . .	17
2.6	Endre flere variabler . . . . .	18
2.7	Heltallsdivisjon . . . . .	19
2.8	Heltallsdivisjon og rest . . . . .	20
2.9	Oppsummering kapittel 2 . . . . .	21
2.10	Oppgaver . . . . .	22
<b>3</b>	<b>Inndata</b>	<b>27</b>
3.1	Lese inn tekst . . . . .	27
3.2	Lese inn tall . . . . .	28
3.3	Lese inn desimaltall . . . . .	30
3.4	Formatert utskrift . . . . .	31
3.5	Prosentkalkulator . . . . .	32
3.6	Oppsummering kapittel 3 . . . . .	33
3.7	Oppgaver . . . . .	34
<b>4</b>	<b>Vilkår og tilfeldighet</b>	<b>39</b>
4.1	Enkel if-setning . . . . .	39

4.2	Tilfeldig heltall . . . . .	40
4.3	Vilkår med if og else . . . . .	41
4.4	If, elif og else . . . . .	42
4.5	Vilkår med or . . . . .	44
4.6	Boolske variabler og and . . . . .	45
4.7	Størst tall . . . . .	46
4.8	Tilfeldig desimaltall . . . . .	47
4.9	Egenkapital og lån . . . . .	48
4.10	Oppsummering kapittel 4 . . . . .	49
4.11	Oppgaver . . . . .	50
<b>5</b>	<b>Løkker</b>	<b>55</b>
5.1	Enkel for-løkke . . . . .	55
5.2	Tallmønstre med addisjon, metode 1 . . . . .	56
5.3	Tallmønstre med addisjon, metode 2 . . . . .	57
5.4	Gange- og delemønstre . . . . .	58
5.5	Summere med løkker . . . . .	59
5.6	While-løkke: sparing . . . . .	60
5.7	Figurtall . . . . .	61
5.8	While True: prøv igjen . . . . .	62
5.9	Oppsummering kapittel 5 . . . . .	63
5.10	Oppgaver . . . . .	64
<b>6</b>	<b>Lister</b>	<b>71</b>
6.1	Elementer i en liste . . . . .	71
6.2	Legge til og slette elementer . . . . .	72
6.3	Sum og lengde . . . . .	73
6.4	Løpe gjennom liste med løkke . . . . .	74
6.5	Løpe gjennom flere lister . . . . .	75
6.6	Algoritme for median . . . . .	76
6.7	Oppsummering kapittel 6 . . . . .	77
6.8	Oppgaver . . . . .	78
<b>7</b>	<b>Funksjoner</b>	<b>89</b>
7.1	Lineær funksjon . . . . .	89
7.2	Funksjoner og løkker . . . . .	90
7.3	Bestemme nullpunkt . . . . .	91
7.4	Eksponentialfunksjon: halveringstid . . . . .	92
7.5	Funksjon med flere parametre . . . . .	93
7.6	Størst mulig overskudd . . . . .	94
7.7	Gjennomsnittlig vekstfart . . . . .	95
7.8	Lage numpy-arrayer . . . . .	96
7.9	Tegne en graf med matplotlib . . . . .	97

7.10	Oppsummering kapittel 7 . . . . .	98
7.11	Oppgaver . . . . .	100
<b>II</b>	<b>Programmering på Vg2</b>	<b>113</b>
<b>8</b>	<b>Logaritmer og likninger</b>	<b>115</b>
8.1	Utregninger med naturlige logaritmer . . . . .	115
8.2	Utregninger med tierlogaritmer . . . . .	116
8.3	Enkel logaritmelikning . . . . .	117
8.4	Logaritmelikning med $\ln$ . . . . .	118
8.5	Eksponentiallikning . . . . .	119
8.6	Halveringsalgoritmen del 1: tallinja . . . . .	120
8.7	Halveringsalgoritmen del 2 . . . . .	121
8.8	Oppsummering kapittel 8 . . . . .	124
8.9	Oppgaver . . . . .	126
<b>9</b>	<b>Grafiske fremstillinger</b>	<b>133</b>
9.1	Linjediagram . . . . .	133
9.2	Linjediagram med pynt . . . . .	134
9.3	Stolpediagram . . . . .	136
9.4	Grafen til en polynomfunksjon . . . . .	138
9.5	Grafene til $e^x$ og $\ln x$ . . . . .	140
9.6	Logaritmisk skala . . . . .	141
9.7	Oppsummering kapittel 9 . . . . .	142
9.8	Oppgaver . . . . .	144
<b>10</b>	<b>Grenseverdier og funksjoner</b>	<b>149</b>
10.1	Grenseverdi der $x$ går mot uendelig . . . . .	149
10.2	Grenseverdi der $x$ går mot 0 . . . . .	150
10.3	Grenseverdi der $x$ går mot et tall . . . . .	152
10.4	Eksponentialfunksjon . . . . .	153
10.5	Numerisk derivasjon . . . . .	154
10.6	Toppunkt . . . . .	155
10.7	Ikke-kontinuerlig funksjon . . . . .	156
10.8	Newtons metode del 1 . . . . .	158
10.9	Newtons metode del 2 . . . . .	160
10.10	Oppsummering kapittel 10 . . . . .	162
10.11	Oppgaver . . . . .	164
<b>11</b>	<b>Sannsynlighet (S1)</b>	<b>169</b>
11.1	To like terningkast . . . . .	169
11.2	Simulere fødsler . . . . .	170

11.3	Sum av terningkast . . . . .	172
11.4	Trekke kuler . . . . .	173
11.5	Trekke kuler fra to esker . . . . .	174
11.6	Trekning til et styre . . . . .	175
11.7	Skiskyting . . . . .	176
11.8	Oppslutning og meningsmåling . . . . .	177
11.9	Vinnertall i Lotto . . . . .	178
11.10	Oppsummering kapittel 11 . . . . .	179
11.11	Oppgaver . . . . .	182
<b>12</b>	<b>Reelle data og modellering</b>	<b>193</b>
12.1	Orkaner . . . . .	193
12.2	Lese data fra fil . . . . .	194
12.3	To bedrifter: grafisk fremstilling . . . . .	195
12.4	To bedrifter: dataanalyse . . . . .	196
12.5	Lineær regresjon . . . . .	198
12.6	Polynomregresjon . . . . .	199
12.7	Logaritmisk regresjon . . . . .	201
12.8	Eksponentiell regresjon . . . . .	203
12.9	Oppsummering kapittel 12 . . . . .	205
12.10	Oppgaver . . . . .	208
<b>III</b>	<b>Programmering på Vg3</b>	<b>215</b>
<b>13</b>	<b>Tallfølger og rekker</b>	<b>217</b>
13.1	Tallfølger . . . . .	217
13.2	Tallfølger med rekursiv formel . . . . .	218
13.3	Aritmetisk rekke: rekursivt . . . . .	219
13.4	Aritmetisk rekke: eksplisitt . . . . .	220
13.5	Geometrisk rekke: rekursivt . . . . .	221
13.6	Geometrisk rekke: eksplisitt . . . . .	222
13.7	Uendelige geometriske rekker: rekursivt . . . . .	223
13.8	Uendelige geometriske rekker: eksplisitt . . . . .	224
13.9	Oppsummering kapittel 13 . . . . .	225
13.10	Oppgaver . . . . .	228
<b>14</b>	<b>Integrasjon</b>	<b>237</b>
14.1	Oppdelt areal . . . . .	237
14.2	Trapez og rektangler . . . . .	238
14.3	Arealet under grafen . . . . .	240
14.4	Arealet under en parabel . . . . .	242
14.5	Rektangelsum over $\ln$ . . . . .	244



14.6	Negative integraler . . . . .	246
14.7	Arealet mellom to grafer . . . . .	248
14.8	Oppsummering kapittel 14 . . . . .	250
14.9	Oppgaver . . . . .	252
<b>15</b>	<b>Diskrete sannsynlighetsfordelinger (S2)</b>	<b>263</b>
15.1	Stokastisk variabel . . . . .	263
15.2	Utforske forventningsverdien . . . . .	264
15.3	Simulere binomisk sannsynlighet . . . . .	266
15.4	Utforske og beregne variansen . . . . .	268
15.5	Sannsynlighetsberegning binomisk . . . . .	270
15.6	Simulere hypergeometriske forsøk . . . . .	272
15.7	Hypergeometrisk fordeling . . . . .	274
15.8	Oppsummering kapittel 15 . . . . .	276
15.9	Oppgaver . . . . .	278
<b>16</b>	<b>Kontinuerlige sannsynlighetsfordelinger (S2)</b>	<b>285</b>
16.1	Kontinuerlig stokastisk variabel . . . . .	285
16.2	Ventetid . . . . .	287
16.3	Normalfordelingen: simulering . . . . .	289
16.4	Normalfordelingen: grafisk . . . . .	291
16.5	Standardnormalfordeling: kalkulator . . . . .	293
16.6	Sentralgrensesetningen: plott . . . . .	294
16.7	Sentralgrensesetningen: simulering . . . . .	296
16.8	Enkel hypotesetest: binomisk . . . . .	299
16.9	Hypotesetest på reelle data . . . . .	301
16.10	Tosidig test . . . . .	303
16.11	Oppsummering kapittel 16 . . . . .	305
16.12	Oppgaver . . . . .	308
	<b>Tillegg</b>	<b>317</b>
<b>A</b>	<b>Referanseprogrammer i R1</b>	<b>319</b>
A.1	Tegne grafen til den omvendte funksjonen . . . . .	319
A.2	Undersøke om to vektorer er like . . . . .	319
A.3	Lengden av vektorer . . . . .	319
A.4	Vektorsum manuelt . . . . .	320
A.5	Vektorsum med numpy . . . . .	320
A.6	Multiplikasjon av vektor med skalar (tupple) . . . . .	320
A.7	Multiplikasjon av vektor med skalar (liste) . . . . .	320
A.8	Avgjøre parallellitet . . . . .	321
A.9	Skalarprodukt (prikkprodukt) . . . . .	321
A.10	Avgjøre ortogonalitet ved skalarprodukt . . . . .	321

A.11	Vinkelen mellom to vektorer med numpy . . . . .	322
A.12	Parameterfremstilling gitt to punkter . . . . .	322
A.13	Parameterfremstilling av kurve grafisk . . . . .	323
<b>B</b>	<b>Referanseprogrammer i R2</b>	<b>325</b>
B.1	Konvertere fra grader til radianer . . . . .	325
B.2	Konvertere fra radianer til grader . . . . .	325
B.3	Vinkel i første omløp (grader) . . . . .	325
B.4	Vinkel i første omløp (radianer) . . . . .	326
B.5	Numerisk integrasjon med trigonometrisk funksjon . . . . .	326
B.6	Numerisk integrasjon: omdreiningslegeme . . . . .	327
B.7	Punkt inni kule . . . . .	327
B.8	Skalarprodukt (prikkprodukt) i 3D . . . . .	328
B.9	Vektorprodukt (kryssprodukt) i 3D . . . . .	328
B.10	Lengden av en vektor . . . . .	328
B.11	Volumprodukt . . . . .	329
B.12	Avstand fra punkt til plan . . . . .	329
B.13	Tegne retningsdiagram . . . . .	330
B.14	Tegne omdreiningslegeme i 3D . . . . .	331
<b>C</b>	<b>Installering av Python-biblioteker</b>	<b>333</b>
C.1	Generell framgangsmåte i terminal . . . . .	333
C.2	Installere bibliotek gjennom Thonny . . . . .	333
<b>D</b>	<b>Nettressurser</b>	<b>335</b>
D.1	Hovedsiden på GitHub . . . . .	335
D.2	Finne løsninger . . . . .	335
D.3	Finne csv-filer . . . . .	335
D.4	Kartlegging . . . . .	336
D.5	Ekstra stoff . . . . .	336
D.6	Diskusjonsforumet . . . . .	336
D.6.1	Velg og lese diskusjoner . . . . .	336
D.6.2	Opprette diskusjon . . . . .	336
D.6.3	Formatering i diskusjoner . . . . .	338
D.7	Melde feil i boka . . . . .	338
D.8	Feil eller forbedring i kode . . . . .	339
<b>E</b>	<b>Til Læreren</b>	<b>341</b>
E.1	Innhold . . . . .	341
E.2	Implementere programmering i undervisningen . . . . .	341
E.3	Kort om Python . . . . .	341
E.4	Fremdriftsplaner . . . . .	342
E.4.1	Fremdriftsplan Vg1 . . . . .	342

E.4.2	Fremdriftsplan Vg2 . . . . .	342
E.4.3	Fremdriftsplan Vg3 . . . . .	342
E.5	Nivådifferensiering . . . . .	343
E.6	Oppgavetyper: oversikt og analyse . . . . .	343
E.6.1	Kode-rekkefølge . . . . .	343
E.6.2	Kode med feil . . . . .	344
E.6.3	Kodepuslespill . . . . .	344
E.6.4	Algoritme på norsk . . . . .	344
E.6.5	Skjelettkode . . . . .	345
E.6.6	Tekst . . . . .	345
E.7	Utvikling av programmeringskunnskap i fem faser . . . . .	346
E.8	Opplæringens struktur: PRIMM . . . . .	347
E.9	PRIMM-analyse: et konkret eksempel . . . . .	348
<b>Figurer</b>		<b>349</b>
<b>Tabeller</b>		<b>354</b>
<b>Python</b>		<b>355</b>
<b>Stikkordsregister</b>		<b>357</b>
<b>Bibliografi</b>		<b>368</b>



## **DEL I**

---

# **Programmering på Vg1**

---



# KAPITTEL 1

## Skrive til skjerm

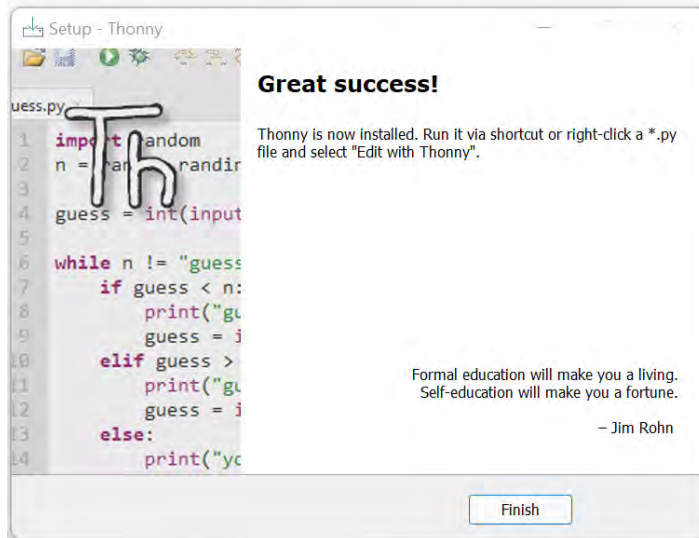
### 1.1 Nedlasting av Thonny



Figur 1.1: Nedlastingsvalg på nettsiden [thonny.org](https://thonny.org).

Åpne nettsiden [thonny.org](https://thonny.org). Du vil se nedlastingsalternativene øverst på nettsiden, som illustrert i figur 1.1. Velg versjonen som er kompatibel med ditt operativsystem (Windows, Mac eller Linux), og vent på at nedlastingen fullføres.

### 1.2 Installasjon av Thonny



Figur 1.2: Installasjonen er ferdig.

Åpne den nedlastede installasjonsfilen. Følg veiviseren ved å klikke på **Next** flere ganger. Klikk deretter på **Install** for å igangsette installasjonen. Klikk på **Finish** etter at installasjonen er fullført. Se figur 1.2.

## 1.3 Skriv ut «Hallo, Norge»

(0103\_hallo.py)

Du skal nå få datamaskinen til å skrive ut teksten `Hallo, Norge` til skjermen.

- Åpne programmet Thonny.
- Skriv koden `print("Hallo, Norge")` i den øvre delen av Thonny. Se figur 1.3.




Figur 1.3: Første program i Thonny.

Det er vanlig å gjøre feil når man programmerer. La oss se på noen typiske feil:



<code>print"Hallo, Norge")</code>	<code>pritrn("Hallo, Norge")</code>	<code>print(Hallo, Norge)</code>
<code>print("Hallo, Norge')</code>	<code>Print("Hallo, Norge")</code>	<code>print("Hallo, Norge)</code>

Insektet symboliserer at det er feil («bugs») i koden.

- Bestem feilen i hver av kodelinjene ovenfor.
- Rett eventuelle feil i koden din.
- Fra menyen velg `File > Save as...`, og lagre som «0103\_hallo.py».
- Klikk på  for å kjøre programmet. Nå bør teksten `Hallo, Norge` vises i den delen av Thonny som kalles Shell. Se figur 1.3.

Gratulerer! Du har nå laget bokas første Python-program.



Boka legger opp til bruk av Thonny som Python-editor, men du står selvsagt fritt til å velge andre programmer som for eksempel «Visual Studio Code» eller «Spyder».



## 1.4 Skrive ut et dikt



(0104\_dikt.py)

```

1 print("Likevel rommer det")
2 print("")
3 print("hele evigheten.")

```

Python-kommandoen `print` skriver ut tekst eller verdier til skjermen.

- Les koden ovenfor, og gjett på resultatet.
- Lag en ny fil ved å klikke på , eller velg **File** > **New** fra menyen.
- Skriv av koden, lagre som «0104\_dikt.py» og klikk på  for å kjøre programmet.
- La kodelinjene 1 og 2 bytte plass. Gjett på resultatet, før du kjører programmet.
- Legg til kodelinja `print("Nå.")` nederst i koden.
- Legg til kodelinja `print("Et meget lite ord")`.

```

Nå.

Et meget lite ord.
Nå.
Likevel rommer det
hele evigheten.

```

- Skriv ferdig koden slik at resultatet blir som vist ovenfor.<sup>1</sup>



Figur 1.4: Hans Børli's poetiske verden: Hytte i skogen.


<sup>1</sup>Dikt av Hans Børli.[1]

## 1.5 Regne med Python

(0105\_regne.py)

```
1 print(3 + 4)
2 print("3 + 4")
```

Python kan utføre regneoperasjoner for deg.

- Les koden ovenfor, og gjett på resultatet.
- Lag en ny fil ved å klikke på , eller velg **File** > **New** fra menyen.
- Skriv av koden, lagre som «0105\_regne.py» og kjør programmet.
- Legg til kodelinja `print(12 - 4*2)`.
- Legg til kodelinja `print("12 - 4*2")`. Gjett på resultatet, før du kjører programmet.
- Legg til den riktige kodelinja som regner ut  $100 - 25 \div 3$ , og skriver svaret til skjermen:

```
100 - 25/3          print(100 - 25/3)          print("100 - 25/3")
```

- Les koden nedenfor, og gjett på resultatet.

```
print("(10-3/6) * 2 =")
print((10-3/6) * 2)
```

- Legg til kodelinjene ovenfor, og kjør programmet.
- Legg til kode som regner ut og skriver ut svaret på regnestykkene  $2 \cdot 5 - 10$  og  $(20 - 3 \div 12) \cdot (2 + 4 \cdot 21)$ . Resultatet av dette skal være slik:

```
2*5 - 10 =
0
(20 - 3/12) * (2 + 4*21) =
1698.5
```




Figur 1.5: Snart snakker du pythonsk.

## 1.6 Potenser

(0106\_potenser.py)

```
1 print(2*3)
2 print(2**3)
3 print(2*4)
4 print(2**4)
```

En potens, som for eksempel  $2^4 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2}^4 = 16$ , skrives i Python som `2**4`.

- Les koden ovenfor, og gjett på resultatet.
- Lag en ny fil ved å klikke på , eller velg **File** > **New** fra menyen.
- Skriv av koden, lagre som «0106\_potenser.py» og kjør programmet.
- Endre kodelinje 2 til `print(2**3*2)`. Gjett på resultatet, før du kjører programmet.
- Legg til en kodelinje for å regne ut  $10^3$ . Kontroller at svaret blir 1000.
- Endre kodelinje 4 til `print("2**4 =", 2**4)`.
- Legg til kodelinja nedenfor som gir resultatet `3**4 = 81`.

```
print("3**4 =", 3**4)      print(3**4)
print("3*4 =", 3**4)      print("3**4=", 3**4)
```

- Utvid programmet slik at det regner ut  $4^5 - 3^6$  og resultatet presenteres slik:

```
4**5 - 3**6 = 295
```




Figur 1.6: Å regne på gamlemåten.

## 1.7 Kvadratrøtter

(0107\_rotter.py)

```
1 import math
2 print(math.sqrt(4))
3 print(math.sqrt(25))
```

Koden `math.sqrt(25)` regner ut kvadratroten til 25, det vil si  $\sqrt{25} = 5$ .

- Les koden ovenfor, og gjett på resultatet.
- Lag en ny fil ved å klikke på , eller velg **File** **>** **New** fra menyen.
- Skriv av koden, lagre som «0107\_rotter.py» og kjør programmet.
- Endre på kodelinje 3 for å regne ut  $\sqrt{36}$ .
- Legg til den korrekte av kodelinjene nedenfor som regner ut  $\sqrt{125 - 25} \cdot 2$ .

```
print(math.sqrt(125-25)**2)
```

```
print(math.sqrt(125-25)*2)
```

```
print(math.Sqrt(125-25)*2)
```

```
print(math:sqrt(125-25)*2)
```

Resultatet av kodelinja skal bli `20.0`.

I matematikken på VGS lærer du at det ikke er mulig å ta kvadratroten til et negativt tall. For eksempel finnes ikke tallet  $\sqrt{-5}$ .

- Legg til kode for å regne ut  $\sqrt{-5}$ . Undersøk hva slags feilmelding du får. Endre deretter tallet  $-5$  til  $5$ , for å unngå feilmeldingen.
- Legg til kodelinja nedenfor. Gjett på resultatet, før du kjører programmet.

```
print("Kvadratroten til 144 er", math.sqrt(144))
```

- Fullfør programmet slik at det regner ut stykkene nedenfor, og skriver svarene på skjermen. Resultatet skal være som vist til høyre.

(i)  $\sqrt{1234 \cdot 5}$

78.54934754662193

(ii)  $\frac{\sqrt{6^2 + 8^2}}{2}$

5.0



For å få større knapper i Thonny, velg **Tools** **>** **Options...** **>** **General** **>** **UI scaling factor** i menyen. Sett «UI scaling factor» til for eksempel 2,5, og start Thonny på nytt.

## 1.8 Oppsummering kapittel 1

I dette kapitlet lærte du å bruke `print` -kommandoen til å skrive ut tekst og utføre regneoperasjoner.

### Utskrift av tekst

<code>print("Hei")</code>	Hei
<code>print("")</code>	
<code>print("på deg.")</code>	på deg.

### Regneoperasjoner

<code>print("5*3 + 4")</code>	5*3 + 4
<code>print(5*3 + 4)</code>	19
<code>print(3**2 - 10/2)</code>	4.0
<code>print("3/4 = ", 3/4)</code>	3/4 = 0.75

### Kvadratrot

<pre>import math print(math.sqrt(9)) print("Kvadratroten til 120 er", math.sqrt(120))</pre>	
3.0	
Kvadratroten til 120 er 10.954451150103322	

### Feilsøking

<code>Print(5+3*4)</code>	<code>NameError: name 'Print' is not defined</code>
---------------------------	---

Python gjenkjenner ikke `Print` som en gyldig kommando fordi den er skrevet med stor P.

### Perfeksjon

For at et Python-program skal kjøre som normalt, må det være helt feilfritt!

## 1.9 Oppgaver

### Oppgave 1.1A

(opg\_0101A.py)



```
Print("Mount Everest")
print(er verdens høyeste)
print"fjell.")
```

Skriv av koden, og rett opp feilene slik at programmet kjører uten feilmelding.

### Oppgave 1.2A

(opg\_0102A.py)

```
print("Elon Musk")
...
...
```

```
Elon Musk
er
rik.
```

Skriv ferdig koden slik at den produserer det angitte resultatet.

### Oppgave 1.3A

(opg\_0103A.py)

```
7.0
12
4
0
```

```
print(3*4)      print(100-99-1)
print(42/7 + 1) print(7*4-6*4)
```

Sett sammen kodelinjene i riktig rekkefølge slik at resultatet av programmet blir som vist ovenfor.

### Oppgave 1.4A

(opg\_0104A.py)

Lag et program som regner ut stykkene nedenfor, og skriver ut svarene.

- (i)  $10 \cdot 3 - 7$                       (ii)  $(20 - 8 \div 2) \cdot 2$   
(iii)  $3^5 + 2^{10}$                         (iv)  $10^{46-7^2}$

### Oppgave 1.5A

(opg\_0105A.py)

Lag et program som regner ut  $\frac{123 \cdot 234}{321 \cdot 432}$ . Resultatet skal vises som følger:

```
(123*234) / (321*432) = ...
```

## 1. Skrive til skjerm

### Oppgave 1.6B

(opg\_0106B.py)

Lag et program som regner ut  $\sqrt{3^2+4^2}$  og  $\sqrt{\left(\frac{1}{2}\right)^{-8}}$ .

### Oppgave 1.7B

(opg\_0107B.py)

I matematikk tar vi også i bruk tredjerot, fjerderot og så videre, i tillegg til kvadratrot. Følgende sammenhenger gjelder:

$$\sqrt{16} = 16^{1/2} = 4$$

$$\sqrt[3]{27} = 27^{1/3} = 3$$

$$\sqrt[4]{16} = 16^{1/4} = 2$$

Bruk dette til å lage et program som regner ut følgende, uten bruk av `math.sqrt` :

(i)  $\sqrt{16}$

(ii)  $\sqrt[4]{16}$

(iii)  $\sqrt[3]{1337}$

(iv)  $\sqrt[5]{-1024}$

### Oppgave 1.8B

(opg\_0108B.py)

```
1 print(f"2*5 = {2*5} og 2**5 = {2**5}")
2 ...
3 ...
```

2\*5 = 10 og 2\*\*5 = 32  
10/5 = 2.0 og 10-5 = 2  
10\*\*5-9\*\*5 = 40951

Bruk av `print(f"...")` lar oss kombinere tekst med utregning av matematiske uttrykk. Uttrykket vi ønsker å regne ut plasseres inni `{...}`. Skriv ferdig koden ovenfor slik at du får resultatet til høyre. Bruk tilsvarende format som på den første kodelinja.

### Oppgave 1.9B

(opg\_0109B.py)

```
1 print(f"Tallet er 123.4726")
2 print(f"1 desimal {123.4726:.1f}")
3 print(f"3 desimaler {123.4726:.3f}")
```

Tallet er 123.4726  
1 desimal: 123.5  
3 desimaler: 123.473

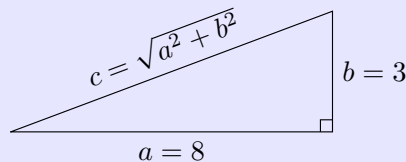
Koden ovenfor viser hvordan vi kan runde av desimaltall. En skulptur er kjøpt for 5000 kr, og verdien øker med 8 % årlig. Etter  $x$  år vil verdien være  $5000 \cdot 1,08^x$ . Lag et program som regner ut verdien hvert år de tre første årene. Presenter resultatet med 2 desimaler slik:

Etter 1 år: ... kr  
Etter 2 år: ... kr  
Etter 3 år: 6298.56 kr

### Oppgave 1.10B 1T

(opg\_0110B.py)

En rettvinklet trekant har sider  $a = 4$ ,  $b = 8$  og  $c$  er den lengste siden (hypotenus). Ifølge Pytagoras' setning er  $a^2 + b^2 = c^2$ . Figur 1.7 viser formelen for  $c$ .



Figur 1.7: Rettvinklet trekant. Ukjent  $c$ .

Lag et program som regner ut og skriver ut  $c$  med 3 desimaler. Se oppgave 1.9 for hvordan du runder av til 3 desimaler i utskriften.

### Oppgave 1.11B 1T

(opg\_0111B.py)

```
1 import math
2 print(math.pi)
```

Volumet av en kule med radius  $r$  er gitt ved

$$V = \frac{4\pi r^3}{3}$$

Ta utgangspunkt i koden ovenfor, og bruk `math.pi` til å beregne volumet av en kule med radius 2,84 m. Avrund svaret til 3 desimaler. Resultatet skal være:

Volumet av en kule med radius 2,84 m:  
95.950 kubikkmeter



### Oppgave 1.12S

(opg\_0112S.py)

Du kan bruke Unicode for å produsere mer estetisk tiltalende utskrift i Python. For å skrive ut  $3^2$ , må du skrive `print(u"3\u00b2")`, der `\u00b2` er unicode-tallet for  $^2$ . Merk at `print`-kommandoen må begynne slik: `print(u"...")`. Finn flere unicode-tall på nettsiden <https://unicode-table.com/en/sets/superscript-and-subscript-letters/>, og lag et program som skriver ut følgende:

$10^4 + 5^{2+6-9}$



## **DEL II**

---

# **Programmering på Vg2**

---



# KAPITTEL 8

## Logaritmer og likninger

### 8.1 Utregninger med naturlige logaritmer (0801\_natlog.py)

```
1 import numpy
2
3 e = numpy.e # Eulers tall
4 ln = numpy.log # Naturlig logaritme
5 print(e)
6 print(ln(e))
```

Vi skal benytte oss av biblioteket *numpy* til å utføre beregninger med logaritmer.

- a) Les koden ovenfor, og gjett på resultatet.
- b) Skriv av koden, lagre som «0801\_natlog.py» og kjør programmet.



Dersom du får en feilmelding ved kjøring av programmet, er det mulig at *numpy* ikke er installert. Se side 333 for en gjennomgang av installeringen.

- c) Legg til kodelinja `print(ln(e**3))` for å regne ut  $\ln e^3$ .

For å vise både uttrykket og svaret, kan vi bruke formen `print(f"{...} = ")`.

- d) Endre kodelinje 5-7 slik:

```
ln = numpy.log
print(f"{e} = ")
print(f"{ln(e)} = ")
print(f"{ln(e**3)} = ")
```

- e) Legg til en kodelinje for å regne ut  $\ln e^6$ .
- f) Legg til kodelinja `print(ln(e**(-2)))` for å regne ut  $\ln e^{-2}$ .
- g) Legg til en kodelinje for å regne ut  $\ln e^{-9}$ .

Kodelinja `print(ln(15+3**5)-e**2)` regner ut  $\ln(15 + 3^5) - e^2$ .

- h) Legg til kode som regner ut  $2^3 - \ln(e^4 + 50)$ .

## 8.2 Utregninger med tierlogaritmer

(0802\_tierlog.py)

```
1 import numpy
2
3 lg = numpy.log10
4 print(lg(100))
```

Vi kan beregne tierlogaritmer ved hjelp av numpy sin `log10`-kommando. I koden ovenfor har vi lagret denne kommandoen ved navnet `lg`.

- a) Les koden ovenfor, og gjett på resultatet.
- b) Skriv av koden, lagre som «0802\_tierlog.py» og kjør programmet.
- c) Legg til kodelinja `print(lg(10**4))` for å regne ut  $\lg 10^4$ .

```
1 import numpy as np
2
3 lg = np.log10
4 print(lg(100))
```

Ved å legge til koden `as np` på den første kodelinja, kan vi bruke `np` i stedet for `numpy` videre i koden.

- d) Endre koden som vist ovenfor.
- e) Legg til kodelinja `print(10**(lg(0.5)))` for å regne ut  $10^{\lg 0.5}$ .
- f) Legg til en kodelinje for å regne ut  $10^{\lg 23}$ .
- g) Legg til kodelinja `print(f"{lg(3*4)} = ")` for både å vise regnestykket og å vise svaret.
- h) Legg til en kodelinje for å regne ut  $\lg 3 + \lg 4$ . Resultatet skal se slik ut:

```
lg(3) + lg(4) = 1.0791812460476249
```

Sammenlign svaret med g).

- i) Bruk en `for`-løkke for å regne ut  $\lg 10^i$  der  $i \in \{-3, -2, -1, 0, 1, 2, 3\}$ . Nøyaktig to av følgende kodelinjer er nødvendige for å få til dette:

```
print(lg(10**i))      for i in range(-3, 4):
print(lg(i**10))      print(lg(i))
```

## **DEL III**

---

# **Programmering på Vg3**

---



# KAPITTEL 13

## Tallfølger og rekker

### 13.1 Tallfølger

(1301\_folger.py)

```
1 a = 3
2 print(a)
3 a = a + 4
4 print(a)
5 a = a + 4
6 print(a)
```

Vi begynner med å lage et program for å utforske tallfølgen  $\{a_n\} = \{3, 7, 11, \dots\}$ .

- Les koden ovenfor, og gjett på resultatet.
- Skriv av koden, lagre som «1301\_folger.py» og kjør programmet.

Vi bruker `for` - eller `while` -løkker for å gjenta kodelinjer.

- Endre koden som vist nedenfor.

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

- Endre `print` -kommandoene inni `for` -løkka til `print(a, end=" ", " ")`.
- Endre koden slik at resultatet blir som vist nedenfor.

```
3, 7, 11, 15, 19
```



Du trenger bare å endre antall gjentakelser av `for` -løkka.

- Endre koden slik at resultatet blir som vist nedenfor.

```
3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63,
67, 71, 75, 79, 83, 87, 91, 95, 99, 103, 107, 111, 115, 119,
123, 127, 131, 135, 139, 143, 147, 151, 155, 159, 163, 167,
171, 175, 179, 183, 187, 191, 195, 199, 203, 207
```

### 13.3 Aritmetisk rekke: rekursivt

(1303\_arit1.py)

```

1 a = 4
2 for i in range(5):
3     print(a)
4     a += 3

```

Vi skal utforske *aritmatiske* tallfølger som for eksempel 4, 7, 10, 13, ..., men denne gangen er vi interessert i summen av tallene. Da uttrykker vi tallfølgen som en *rekke*:

$$4 + 7 + 10 + 13 + \dots$$

- Les koden ovenfor, og gjett på resultatet.
- Skriv av koden, lagre som «1303\_arit1.py» og kjør programmet.
- Endre `print`-kommandoen til `print(a, end=" ")`.
- Endre `for`-løkka slik at resultatet blir 4 7 10 13 16 19 22.
- Endre koden slik at resultatet blir 2 5 8 11 14 17 20.
- Endre koden slik at resultatet blir 2 10 18 26 34 42 50.
- Legg til følgende kodelinjer på riktig plass slik at du beregner summen  $2 + 10 + 18 + 26 + 34 + 42 + 50$ :

```

summ = a
print(f"Summen er {summ}") summ += a

```

Bare én av kodelinjene skal plasseres inni `for`-løkka. Kontroller at resultatet blir Summen er 182.

- Endre koden slik at du beregner summen:

$$2 + 10 + 18 + \dots + 186$$



Figur 13.1: Trapper til himmelriket. Fantasi.



---

**Tillegg**

---



# TILLEGG A

## Referanseprogrammer i R1

### A.1 Tegne grafen til den omvendte funksjonen

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(-2, 6, 400)
5 f = 4*x - 2
6
7 plt.plot(x, f, label='f(x)')
8 plt.plot(f, x, label='Invers av f(x)')
9 plt.legend()
10 plt.grid(True)
11 plt.show()
```

Programmet tegner grafen til  $f(x) = 4x - 2$  for  $x \in [-2, 6]$  sammen med den omvendte funksjonen til  $f$ .

### A.2 Undersøke om to vektorer er like

```
1 u = (3, 5) # Eller u = [3, 5]
2 v = (3, 5) # Eller v = [3, 5]
3 if u == v:
4     print("u er lik v")
```

Programmet lagrer vektorene  $\vec{u} = [3, 5]$  og  $\vec{v} = [3, 5]$  og undersøker om de er like.

### A.3 Lengden av vektorer

```
1 u = (3, -4)
2 lengde = (u[0]**2 + u[1]**2) ** 0.5
3 print(f"Lengden av vektoren {u} er {lengde:.2f}")
```

Programmet bruker Pytagoras' setning til å beregne lengden av vektoren  $\vec{u} = [3, -4]$ .

## TILLEGG *B*

# Referanseprogrammer i R2

### B.1 Konvertere fra grader til radianer

```
1 import numpy as np
2
3 grader = 360
4 radianer = np.radians(grader)
5
6 print(f"{grader} grader er lik {radianer:.2f} radianer.")
7 # 6.28 radianer
```

Programmet konverterer fra  $360^\circ$  til radianer, og bekrefter at det blir  $2\pi \approx 6,28$ .

### B.2 Konvertere fra radianer til grader

```
1 import numpy as np
2
3 radianer = np.pi / 2
4 grader = np.degrees(radianer)
5
6 print(f"{radianer:.2f} radianer er lik {grader} grader.")
7 # 90.0 grader
```

Programmet konverterer fra  $\pi/2$  radianer til grader.

### B.3 Vinkel i første omløp (grader)

```
1 import numpy as np
2
3 v_grader = 1500
4 v_1omlop = v_grader % 360
5 print(v_1omlop) # 60
```

Programmet bestemmer vinkelen  $v = 1500^\circ$  i første omløp ved å bruke modulooperatoren `%` (som gir resten i en divisjon).

# *TILLEGG E*

## *Til Læreren*

### **E.1 Innhold**

I dette tillegg finnes informasjon som er spesielt nyttig for matematikklærere.

### **E.2 Implementere programmering i undervisningen**

«Hvordan implementerer jeg programmering i matematikkundervisningen?», og «hvordan finner jeg tid til det?» Begge spørsmålene er høyst aktuelle. Her er noen tips til implementering og tidsbruk:

- **Variasjon:** Når du merker at elevene har behov for variasjon, for eksempel etter en lengre oppgaveøkt, kan du la elevene programmere 15 til 30 minutter.
- **Lekse:** Innlæringsdelen i Kaares kokebok er laget for at flest mulig skal kunne følge den med minimal lærerveiledning. Den egner seg derfor godt som hjemmearbeid.
- **Tidsbruk:** Et ukentlig tidsbruk på mellom 15 og 30 minutter (inkludert lekser) skal gi god margin til å komme gjennom kapitlene i god tid før standpunkt/eksamen til våren, både for Vg1, Vg2 og Vg3. Vi anbefaler en jevnlig ukentlig dose, framfor sporadiske heldagsøkter.
- **Nytt stoff:** Når elevene begynner med et nytt kapittel i Kaares kokebok, anbefaler vi at dette skjer i klasserommet. Da får vi muligheter til å rydde opp i typiske feil eller misforståelser tidlig, i stedet for at elevene sitter hjemme og ikke kommer noen vei. Det er opp til læreren om han ønsker å gjennomgå noe felles, eller bare be elevene gjøre for eksempel 2.1-2.3 og deretter gå rundt og hjelpe elevene.

### **E.3 Kort om Python**

Python er et programmeringsspråk skapt av Guido van Rossum i 1991. Python er et av de mest populære språkene for programmering i realfag på universiteter og høyskoler. Den enkle syntaksen gjør språket enkelt å lese og forstå, og dermed velegnet for nybegynnere.

## E.4 Fremdriftsplaner

Her kommer forslag til fremdriftsplaner i Vg1, Vg2 og Vg3. Planene forutsetter et ukentlig tidsbruk på mellom 15 til 30 minutter.

### E.4.1 Fremdriftsplan Vg1

Måned	Sep	Okt	Nov	Des	Jan	Feb	Mars	April
Kapitler	1, 2	2, 3	3, 4	4, 5	5, 6	6, 7	7	rep.

Tabell E.1: Fremdriftsplan Vg1.

### E.4.2 Fremdriftsplan Vg2

Måned	Sep	Okt	Nov	Des	Jan	Feb	Mars	April
Kapitler	8	8, 9	9, 10	10, 11	11	12	12	rep.

Tabell E.2: Fremdriftsplan Vg2 S1.

For R1 er ikke kapittel 11 om sannsynlighet relevant. I denne perioden kan R1-klasser bruke tid på referanseprogrammene til R1 på side 319.

### E.4.3 Fremdriftsplan Vg3

Måned	Sep	Okt	Nov	Des	Jan	Feb	Mars	April
Kapitler	13	13, 14	14	14, 15	15	16	16	rep.

Tabell E.3: Fremdriftsplan Vg3 S2.

For R2 er ikke kapitlene 15 og 16 om sannsynlighet relevante. I denne perioden kan R2-klasser bruke tid på referanseprogrammene til R2 på side 325. Enkelte elever kan utfordres med å tegne Riemann-summer med matplotlib. Dette kapitlet finnes blant våre nettressurser på [github.com/jdforlag/kaares\\_kokebok\\_vgs](https://github.com/jdforlag/kaares_kokebok_vgs) i mappen tillegg. På samme sted ligger et hefte om symbolske beregninger med sympy: De mest ivrige elevene kan lære seg å bruke sympy for å utføre algebraiske manipulasjoner, regne med symboler, og regne eksakt.

## E.5 Nivådifferensiering

Boka er bygget opp for å gjøre nivådifferensiering enkelt. Her kommer et konkret eksempel der vi ser på fire ulike kompetansenivåer.

Kompetansenivå	Aktuelt kapittelinnhold
Lavt	Det meste av innlæringsdelen og de 2-3 enkleste A-oppgavene.
Middels	Hele innlæringsdelen, de fleste A-oppgaver og en B-oppgave eller to etter behov.
Høyt	Hele innlæringsdelen, kan hoppe over A-oppgaver som er for enkle og fokusere på å gjøre B-oppgaver.
Ekstra høyt	Hele innlæringsdelen, men for mange elever på dette nivået vil det være tilstrekkelig å bla gjennom den. Eleven går snarest mulig i gang med B- og S-oppgaver.

Tabell E.4: Kompetansenivåer og aktuelt kapittelinnhold.

## E.6 Oppgavetyper: oversikt og analyse

De forskjellige oppgavetyperne i boka fungerer ikke bare som variasjon. De øver spesifikke ferdigheter. Her følger en oversikt over oppgavetyperne med eksempler.

### E.6.1 Kode-rekkefølge

```
else:
    print("Ikke like")
```

```
kast2 = random.randint(1, 6)
```

```
import random
```

```
if kast1 == kast2:
    print("Like")
```

```
kast1 = random.randint(1, 6)
```

```
print(kast1, kast2)
```

Sett sammen kodebitene på riktig måte. Programmet skal kaste to terninger, skrive ut resultatet, og undersøke om terningene viser like mange øyne.

Oppgavetyperen gir eleven all nødvendig kode for å løse oppgaven, men delt opp i flere kodebiter. Ferdigheten ligger i det å plassere kodebitene på rett måte. Denne oppgavetyperen frigjør eleven fra å huske riktig Python-syntaks, og gir eleven muligheten til å tenke rent algoritmisk.

## E.6.2 Kode med feil



```
For i in range(5, 9)
    print(i)
```

Rett alle feilene i koden slik at programmet fungerer.

Oppgavetypen gir eleven algoritmen i programmet. Oppgaven tester dermed elevenes kunnskap om Python-syntaks.

## E.6.3 Kodepuslespill

```
1 def h(x):
2     return -3*x + 4
3
4 x_verdi = int(input(...))
5 y_verdi = ...
6 print(f"h({x_verdi}) = {...}")
```

y\_verdi

"Oppgi en x-verdi"

h(x\_verdi)

Lag et program der brukeren kan skrive inn en  $x$ -verdi når programmet kjører, og få regnet ut funksjonsverdien til  $h(x) = -3 \cdot x + 4$ . Skriv av koden, og plasser kodebitene til høyre på rett plass der det mangler kode ( ... ).

Oppgavetypen gir eleven all nødvendig kode, men kravet om å flette kodebitene sammen på denne måten krever både syntaktiske og logiske ferdigheter.

## E.6.4 Algoritme på norsk

```
Definer funksjonen  $g(x) = x - 5$ 
Gi a verdien -1
Så lenge  $g(a) < 0$ :
    Øk a med 1
Skriv ut verdien til a
```

Gitt funksjonen  $g(x) = x - 5$ . Vi vet at  $g(-1)$  er negativ og at  $g$  vokser. Lag et program som finner nullpunktet, ved å følge algoritmen ovenfor.

Oppgavetypen gir eleven hele algoritmen som er nødvendig for å løse problemet. Oppgaven tester dermed elevens ferdigheter med Python.



### E.6.5 Skjelettkode

```

1  tall = int(input(...))
2  printall = True
3  for deletall in range(2, tall):
4      # Hvis tall kan deles på deletall
5      # Sett printall til False
6      # Avslutt løkka
7  if printall:
8      # Skriv ut tallet er et printall
9  else:
10     # Skriv ut tallet ikke er et printall

```

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

Opgavetyper gir elevene algoritmen i form av en ferdig programstruktur, også kalt skjelettkode. Ferdigheten tester både Python-syntaks og logikk; for eksempel å finne ut av hvordan «Hvis tall kan deles på deletall» skal løses i Python.

### E.6.6 Tekst

Gitt funksjonen

$$f(x) = \frac{2x-3}{4-x}, \quad D_f = \langle 4, \infty \rangle$$

Skriv et program som undersøker hva som skjer med  $f$  når  $x$  går mot uendelig. I praksis kan du gjøre dette ved skrive ut  $f(5), f(10), f(20), f(40)$  og så videre. Begynn med 10 iterasjoner i løkka, øk deretter til 60. Resultatet skal være:

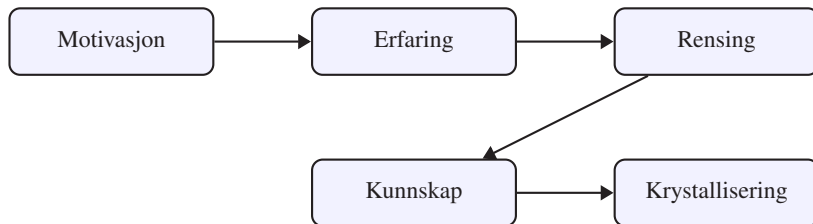
```

-7.0
-2.8333333333333335
-2.3125
...

```

Opgavetyper gir eleven minimalt eller ingen hjelp med hverken algoritme eller Python-syntaks. Eleven får dermed prøvd sine ferdigheter i både algoritmisk tenkning og sin evne til å produsere egen kode.

## E.7 Utvikling av programmeringskunnskap i fem faser



Figur E.1: Programmeringskunnskap i fem faser.

Utviklingen av kunnskap i programmering kan deles inn i fem faser.

Den første fasen, betegnet som *motivasjon*, understreker betydningen av elevens engasjement i tilegnelsen av ny kunnskap. Dersom den nye kunnskapen for eksempel er «løkker», kan læreren først, til elevenes sunne frustrasjon, be dem om å skrive ut «Hei» 1000 ganger ved hjelp av like mange kodelinjer. Da vil motivasjonen for forenkling i form av løkker komme naturlig.

Den andre fasen, *erfaring*, innebærer å la elevene få utforske den nye kunnskapen og bli kjent med den gjennom å gjøre en rekke enkle oppgaver. Elevene dykker altså dypere inn i emnet og utvikler en aktiv søken etter forståelse.

I den neste fasen, *rensing*, skal elevene få mulighet til å rydde opp i eventuelle misoppfatninger, og generalisere de nye erfaringene. Nivået på oppgavene er noe høyere.

Når fasen med rensing er gjennomført, har eleven tilegnet seg ny *kunnskap*. Den nye forståelsen er nå forankret på en solid grunnmur.

I den siste fasen, *krystallisering*, skal elevene anvende kunnskapen til å løse oppgaver i kjente og ukjente sammenhenger og situasjoner. Oppgavenes vanskegrad er høy, og den tidligere kunnskapen blir utfordret og utvidet, idet elevene navigerer gjennom komplekse problemstillinger.<sup>1</sup>

Opplæringen i Kaars kokebok er utformet etter mønster av og i tråd med disse fasene.

<sup>1</sup>Salanci, *Didactics of Programming* [10]

## E.8 Opplæringens struktur: PRIMM

Denne boka følger en strukturert tilnærming til opplæring i programmering. Strukturens kortform kalles *PRIMM*<sup>2</sup> og er som følger:

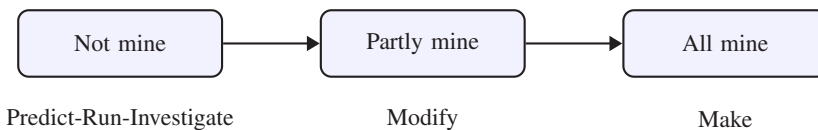
**P**redict (forutsi). Les koden, og forsøk å forutsi hva som blir resultatet. Dette steget er godt egnet for diskusjon i par eller i små grupper.

**R**un (kjør). Skriv av koden, og kjør programmet. Eleven skal forklare eventuelle avvik mellom forventet resultat, og det faktiske resultatet.

**I**nvestigate (undersøk). Gjør små endringer i koden som for eksempel å endre noen verdier eller endre rekkefølgen på kodelinjene. Fortsett med å forutsi hva som vil skje, og kjør deretter programmet på nytt.

**M**odify (endre). Gjør større endringer i koden. Utvid programmet eller lag et liknende program som bygger på dette. Du er nå på vei til å gjøre programmet til ditt eget.

**M**ake (lag). Lag ditt helt eget program.



Figur E.2: PRIMM-diagram.

PRIMM-diagrammet<sup>3</sup> ovenfor viser hvordan elever jobber seg gjennom tre kompetansenivåer i programmering. Først bruker de et program som ikke er deres, leser koden og tester programmet. Deretter gjør de stadig større endringer i koden og lager et program som er delvis deres eget. Til sist skaper de et nytt program som er helt og holdent deres eget produkt. Dette er det høyeste kompetansenivået.

<sup>2</sup>Sentance, [Primm - a structured approach to teaching programming](#) [12]

<sup>3</sup>Sentance, [Exploring pedagogies for teaching programming in school](#) [11]

## E.9 PRIMM-analyse: et konkret eksempel

### Innlæringsdel

```
1 km = 3.8
2 print(f"Du løp {km} km.")
```

Kateryna vil lage et program hvor hun kan registrere distanse i kilometer og tid i timer, og få beregnet gjennomsnittsfarten i km/t og m/s.

- a) Les koden ovenfor, og gjett på resultatet. Predict
- b) Skriv av koden, og kjør programmet. Run
- c) Undersøk hva som skjer dersom du fjerner f-en slik:  
`print("Du løp {km} km.")`. Investigate
- d) Angre endringen og endre tilbake til kodelinje 2 slik det står øverst på siden.
- e) Legg til følgende kodelinjer i riktig rekkefølge for å regne ut gjennomsnittsfarten i km/t. Modify

```
tid_min = 24                                print(f"Farten var {fart_kmt} km/t.")
tid_timer = tid_min / 60                    fart_kmt = km / tid_timer
```

- f) Legg til kode for å regne ut farten i m/s og lagre verdien i variabelen `fart_mps`. Bruk at 1 m/s er 3,6 km/h. Modify
- g) Skriv ut farten i m/s på samme måte som vi skrev ut farten i km/t. Modify

### Oppgave

En kjole koster ordinært 499 kr. Nå selges den med 149 kr avslag. Lag et program som beregner den nye prisen, og skriver den ut. Du må bruke variablene `pris`, `avslag` og `ny_pris` i løsningen. Make

Eksempelen viser hvordan opplæringen i variabler er strukturert etter PRIMM. I denne boka leder innlæringsdelen eleven frem til nivået «Modify», mens det høyeste nivået «Make» fanges opp i oppgavedelen.

# Python

", 4  
""", 92, 154, 224, 297  
\*, 6, 58, 72, 75, 77  
\*\*, 7, 92  
\*\_, 17, 21  
+, 6, 57  
+\_, 17, 21, 74  
-, 6, 57  
-\_, 17, 21  
/, 6, 19, 58  
//, 19, 21, 25, 60  
/\_, 17, 21  
:, 11, 32, 55, 89, 98  
::, 32  
<, 47, 60  
=, 21, 89  
==, 40, 41, 49, 319  
>, 39, 46  
>=, 44, 48, 74  
[], 71, 75, 77  
#, 25, 44  
%, 20, 21, 25  
\_, 48, 60, 238, 250, 280  
\n, 269  
  
abs, 123, 168, 265  
alpha, 292  
alternative, 299  
and, 49, 174  
append, 72, 77  
  
binom, 270  
BinomTestResult, 299  
bins, 281, 291  
break, 84, 247, 273, 309  
  
cmf, 271, 277

color, 140, 291  
combinations, 190  
curve\_fit, Se scipy  
  
data, Se pd  
def, 89, 93, 98  
density, 291  
describe, 301  
  
edgecolor, 291  
elif, 42, 49  
else, 41, 42, 49  
end, 56, 63, 217  
endpoint, 251  
  
f", 11, 15, 21, 31, 89  
False, 45, 49  
figsize, 137  
for, 55, 63, 74, 82, 90  
from, 106, 131, 169  
  
head, 195, 301, 303  
  
if, 39, 49, 74  
import, 8, 25, 40, 96  
in, 74, 82  
index, 190  
inf, 151  
input, 27, 82  
int, 29, 62, 82  
it, 190  
itertools, 190  
  
join, 84  
  
label, 139  
legend, 139  
len, 73, 77  
lg, Se np  
list, 81

ln, Se numpy  
loc, 289  
  
map, 84  
math  
    cos, 112  
    pi, 12, 25, 112  
    sin, 80  
    sqrt, 8, 9, 11, 26  
matplotlib, 97  
    pyplot, Se plt  
max, 75, 77, 84, 199  
mean, 301  
min, 26, 77, 199  
  
NameError, 9  
np, 96, 106  
    arange, 96, 138, 193, 327  
    arccos, 322  
    array, 329  
    cos, 327  
    cross, 328  
    degrees, 322  
    dot, 322  
    e, 115  
    linalg  
        norm, 322  
    linspace, 141, 142, 198  
    log, 115, 142  
    log10, 116, 201  
    mean, 288  
    meshgrid, 330  
    poly1d, 199  
    polyfit, 198  
    random

# Stikkordsregister

## A

1984, 72  
3D, 328  
Aaliyah, 73  
ABC, 177  
Adam, 167  
addisjon, 56, 57  
addsub1, 56  
addsub2, 57  
agn, 45  
agurk, 190  
akselerasjon, 37  
aksenavn, 135  
aksjefond, 168  
aksjeselskap, 175  
alder, 34, 303  
algebra, 25  
algebraisk, 154  
algoritme, 18, 23, 24, 68,  
76, 91, 111,  
158, 224, 229,  
246, 313  
halvering, 120  
Ali, 71  
alternativ form, 204  
Amalie, 76  
analyse, 196  
andregrad, 90  
andregradsfunksjon, 146  
andregradsmodell, 199  
Animal Farm, 33  
anklage, 315  
anleggsvirksomhet, 146  
anmeldelse, 136  
anslå, 175  
antall, 73  
antiderivert, 258  
appendpop, 72  
arbeidsulykke, 146  
areal, 18, 35, 81, 147  
areal mellom, 248  
areal under graf, 240  
arit1, 219  
arit2, 220  
aritmikk, 6  
aritmisk rekke, 219  
asylmottak, 25  
Athena, 299

Audi, 78  
avdeling, 302  
avfyre, Se kanon  
avgift, 47  
avslag, 23  
avsløre, 62  
avstand, 87  
avtale, 65  
avvik, 123  
avvisning, 300

## B

bacon, Se grisekjøt  
bake, 19  
bakeri, 198  
bakken, 107  
bakverk, Se bolle  
ballistikk, Se kanon  
bane, Se kanon  
bank, 48  
Banville, Se Havet  
barn, 170, 181  
batteri, 311  
bedrift, 74, 94, 155  
beep, 82  
befolkningsutvikling, 213  
befrukte, 270  
behandlingsrom, 302  
beherske, 87  
beherskelse, 232  
beholde, 300  
beløp, 60  
Bengt, 19  
benk, 79  
beregne, 6, 22  
beregning, 32  
Bergen, 303  
Bernt, 48  
berolige, 234  
bestemme, 91  
bestige, 281  
besøkende, 79  
betale, 34  
betinget, 40  
bevegelsestype, 54  
bevoktet, 197  
bibliotek, 8, 40, 96, 194  
Big Pharma, 307

bil, 23, 39, 47, 78, 93, 289  
billett, 44, 51, 52  
bilverksted, 14  
binom, 266  
binomialkoeffisient, 270  
binomisk, 176, 180, 277  
binomkalk, 270  
biologisk mann, 303  
Birger, 315  
Bjarne, 93  
bjørk, 78  
blakk, 279  
blindtest, 315  
blod, 92  
blogg, 147  
blueprint, 7  
blyant, 7  
Bo, 175  
bok, 72, 272  
bokklubb, 272  
boks, 109, 291  
bokstav, 77, 179  
boliglån, 48  
bolle, 19  
kanel, 20  
boller, 19  
bom, 176, 279  
boolsk, 45  
bosted, 27  
bot, Se fartsbot  
bra, 42  
bredde, 35  
bruker, 27  
brukertall, 78  
bryllupspar, 304  
bryte, 39, 289  
brød, 198  
brøk, 10, 26  
bug, xv, 4, 55  
bunnpunkt, 139  
buss, 102  
butikk, 184  
byggevirksomhet, 146  
bygård, 35  
bææ, 15  
bølge, 80  
Børli, Se Hans Børli  
Børli, Hans, 5