Tor Espen Kristensen og Christoph Kirfel

Programmering i matematikkfaget

CAPPELEN DAMM AKADEMISK

© CAPPELEN DAMM AS, Oslo, 2022

ISBN 978-82-02-75717-5

1. utgave, 1. opplag 2022
E-utgaven baserer seg på 1. trykte utgave, Cappelen Damm, 2022

Materialet i denne publikasjonen er omfattet av åndsverklovens bestemmelser. Uten særskilt avtale med Cappelen Damm AS er enhver eksemplarfremstilling og tilgjengeliggjøring bare tillatt i den utstrekning det er hjemlet i lov eller tillatt gjennom avtale med Kopinor, interesseorgan for rettighetshavere til åndsverk.

Omslagsdesign: Affair.no
Foto: s. 152: Svetlana Monyakova / Getty Images, s. 179: phi2 / Getty Images, s. 188: Steven White / Getty Images
Sats: Gamma grafisk AS (Vegard Brekke)
Trykk og innbinding: Livonia Print SIA, 2022, Latvia

Boken er utgitt med støtte fra Kunnskapsdepartementet ved Lærebokordningen for høyere utdanning.

www.cda.no
akademisk@cappelendamm.no

Innhold

Forord 7

1 Å komme i gang med Python 9
1.1 Hvorfor Python? 9
1.2 Hva skal jeg installere? 10
1.3 Vi starter med noen enkle utregninger 12
1.4 Variabler i Python 14
1.5 Input fra bruker 17
1.6 Formatering av print 19
1.7 Oppgaver 21

2 Løkker, vilkår og funksjoner 23
2.1 If-tester 23
2.2 For-løkker 25
2.3 While 28
2.4 Funksjoner 30
2.5 Oppgaver 33

3 Datastrukturer 35
3.1 Lister 35
3.2 Mengder 39
3.3 Dictionaries 42
3.4 Moduler i Python 43
3.5 Numpy og arrayer 46
3.6 Rekursiv tenkning 51
3.7 Oppgaver 53

4 Plotting med matplotlib 55
4.1 Plotting av grafen til en funksjon 55
4.2 Stolpediagram 60
4.3 Histogram 64
4.4 Sektordiagram 65
4.5 Subplots og figurer 70
4.6 Oppgaver 74

5 Numeriske metoder 77
5.1 Hvorfor numeriske metoder? 77
5.2 Python og numeriske beregninger 78
5.3 Å finne nullpunkter 79
5.4 Halveringsmetoden 83
5.5 Fikspunktmetoden 86
5.6 Den deriverte til en funksjon 90
5.7 Newtons metode 94
5.8 Numerisk integrasjon 96
5.9 Eulers metode for løsning av differensiallikninger 102
5.10 Oppgaver 104

6 Algoritmer for rotutdragning 109
6.1 Innledning 109
6.2 $\sqrt{2}$ er irrasjonal – et geometrisk resonnement 110
6.3 En algoritme for å regne ut $\sqrt{2}$ 112
6.4 Første generalisering: Vi regner ut $\sqrt{N}$ 117
6.5 Vi optimaliserer algoritmen for urtregning av $\sqrt{N}$ 121
6.6 Newtons metode 124
6.7 Oppsummering 124

7 Databehandling i Python 125
7.1 Lese og skrive til tekstfiler i Python 125
7.2 Pandas – Pythons «regneark» 131
7.3 Kurvetilpasning med regresjon 146
7.4 Behandling av statistiske data 153
7.5 Oppgaver 169

8 Sannsynlighet og statistikk 173
8.1 Simuleringer i Python 173
8.2 Å arbeide med kjente fordelinger 185
8.3 Oppgaver 193

9 CAS med Python 197
9.1 Sympy som CAS-kalkulator 198
9.2 Å jobbe med symboler i Python 200
9.3 Forenkle uttrykk 202
9.4 Løsning av likninger og likningssystem 203
9.5 Plotting av grafer med sympy 210
9.6 Funksjonsdrøfting med sympy 214
9.7 Fra uttrykk til funksjon 218
9.8 Ordinære differensiallikninger 218
9.9 Oppgaver 220

Referanser 223

Forord

Høsten 2020 fikk vi nye læreplaner i norsk skole, og med disse kom programmering inn i skolen for alvor. I rapporten *Teknologi og programmering for alle* (Sanne mfl., 2016), skrevet på oppdrag for Utdanningsdirektoratet, blir det argumentert for at det bør opprettes et eget teknologifag hvor programmering får en sentral plass. Dette rådet ble ikke fulgt. Programmering og algoritmisk tenkning skal være en del av de ulike fagene, der det er naturlig. I arbeidet med de nye læreplanene var det spesielt matematikkfaget som fikk et ansvar for at elevene skulle få kompetanse i programmering og algoritmisk tenkning. Dette er kanskje ikke så unaturlig, siden Utdanningsdirektoratet definerer algoritmisk tenkning som det «å vurdere hvilke steg som skal til for å løse et problem, og å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet».[1] Dette sammenfaller naturlig med kjerneelementet «Utforsking og problemløsing» for matematikkfaget. I beskrivelsen står det blant annet følgende:

> Algoritmisk tenking er viktig i prosessen med å utvikle strategier og framgangsmåter for å løse problemer og innebærer å bryte ned et problem i delproblemer som kan løses systematisk. Videre innebærer det å vurdere om delproblemene best kan løses med eller uten digitale verktøy.[2]

Gjennom denne boken vil du få en solid kompetanse i programmering og algoritmisk tenkning. Vårt mål er at du skal bli i stand til å kunne vurdere ulike strategier og bruke programmering som et nyttig verktøy i problemløsning. Vi forutsetter ikke forkunnskaper i programmering. Boken er bygget opp rundt en mengde eksempler som viser hvordan programmering kan brukes i matematikkfaget for å løse ulike problemer. De ulike eksemplene og oppgavene som er valgt, er relevante for matematikkfaget i skolen.

---

[1] `https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/` (lest: 31.12.21).

[2] Fra kjernelementet *Utforsking og problemløsing*.

Som så mange andre slike bøker har også denne boken sitt utgangspunkt i kompendier laget for studenter. I dette tilfellet har den sitt utspring i et av kursene ved Universitetet i Bergen.[3] Vi vil derfor benytte anledningen til å takke alle tidligere studenter for gode innspill. Vi vil også rette en takk til alle de som har vært involvert i undervisningen i MAT625. Deres innspill har vært verdifulle!

Vi vil spesielt takke Johan Lie, som har kommet med verdifulle bidrag til kapittel 5. Vi vil også takke redaktørene i Cappelen Damm Akademisk for gode innspill og fremragende arbeid med boken.

Boken har en egen nettside. Her vil du finne støttemateriell. Adressen er

```
https://tork73.github.io/
```

[3] https://www.uib.no/math/52932/videreutdanning-lærere

1 Å komme i gang med Python

I dette kapitlet skal du lære

- hvordan du installerer Anaconda
- å lage ditt første program
- hvordan du kjører programmene
- om ulike datatyper og funksjoner knyttet til slike
- å formatere pene utskrifter til skjerm

1.1 Hvorfor Python?

Lingvistene regner med at det er rundt 7000 språk i verden i dag. Vi bruker språket vårt til å formidle mening og innhold. På samme måte må vi ha et språk når vi skal fortelle en datamaskin hva den skal gjøre. Ifølge *The free On-Line Dictionary of Computing* er det rundt 1000 computerspråk.[4] Ett av disse, Python, har vi valgt å bruke i denne boken. Det fins gode grunner for å velge andre språk. Noen vil hevde at Javascript er å foretrekke, siden det kjøres direkte i nettleseren. Når vi likevel velger å bruke Python, er det hovedsakelig av fem grunner. (1) Det er et språk som det er lett å komme i gang med. Du behøver kun å forstå noen enkle prinsipper før du kan lage ditt første program. (2) Det er mange moduler du kan bruke i Python. Dette gjør at du lett kan bruke Python til et bredt spekter av oppgaver, uten å måtte finne opp kruttet på nytt. Du kan tegne grafer, løse likninger, bruke CAS, lage ditt eget brukergrensesnitt etc. (3) Det er i dag mange universiteter som bruker Python for sine nye studenter. (4) Lærebøkene for videregående skole har valgt å bruke Python. (5) Python er et gratis programmeringsspråk med åpen kildekode som er tilgjengelig for alle å bruke. Alt dette gjør at Python i dag er et av de mest brukte programmeringsspråkene.

[4] `http://foldoc.org/contents/language.html`

1.2 Hva skal jeg installere?

Når du skal lage og kjøre et program i Python, er det vanlig å lage programmet i en editor. Det vil si et program som du skriver og lagrer selve koden i. Når du har laget et program som du vil bruke, så må du «kjøre programmet». Det vil si at du må be Python om å lese programmet og gjøre det du ber det om, i koden du har laget.

Selv om det går fint å laste ned Python fra `https://www.python.org/downloads/`, anbefaler vi å laste ned og installere Anaconda. Da vil du i tillegg til Python få med en del biblioteker og editorer du kan bruke. Spyder er en slik editor. Denne gjør det lett å både skrive program og å kjøre koden. Anaconda vil også passe på at du får de riktige versjonene av de ulike tilleggskomponentene som du etter hvert vil bruke. Du laster ned Anaconda fra `https://www.anaconda.com/products/individual`. Se figur 1.1. Når du har lastet ned og installert Anaconda, kan du åpne Anaconda Navigator. Der vil du finne flere programmer som du kan bruke. Velger du å starte Spyder, så er du i gang!



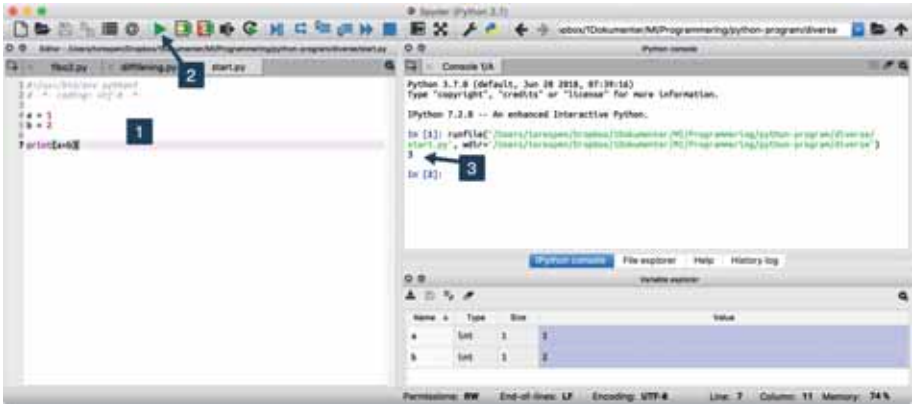
Figur 1.1 Du laster ned Python 3.8 med Anacondas pakkedistribusjon fra Anaconda.com

I feltet til venstre i figur 1.2 har vi skrevet inn følgende kode (1):

```
1 a=1
2 b=2
3 print(a+b)
```

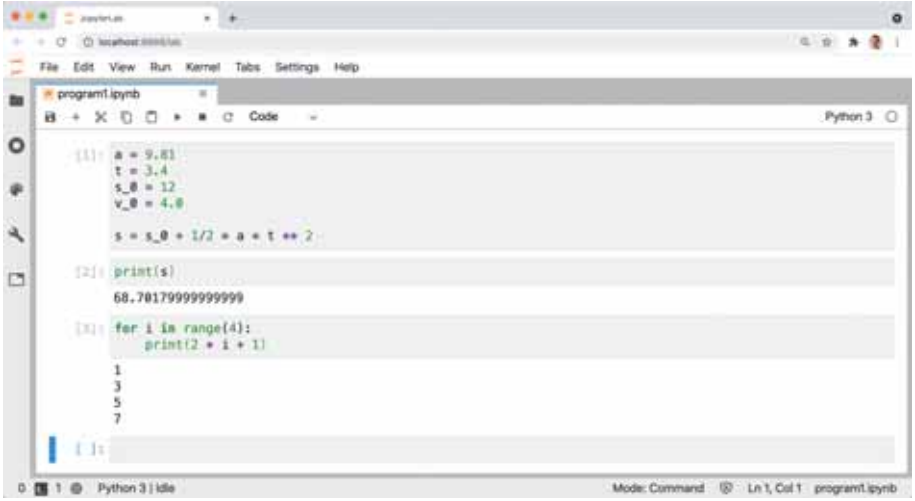
Kode 1.1 En enkel kode som adderer to tall

Når vi trykker på «play»-knappen (2), kjøres koden, og resultatet blir 3, som vist i konsollen (feltet til høyre) (3).



Figur 1.2 Et enkelt program som regner ut $a + b$ når $a = 1$ og $b = 2$

Du finner også et program som heter JupyterLab i Anaconda Navigator. Dette er en webapplikasjon som kjøres i nettleseren din. Figur 1.3 viser hvordan JupyterLab ser ut. JupyterLab er utmerket når du skal demonstrere noe for andre, siden denne viser resultatet av ulike koder på en oversiktlig måte. Du skriver koden inn i de ulike cellene og får resultatet ut nedenfor cellene når du kjører programmet. Skal du gjøre større prosjekter, så er kanskje ikke JupyterLab å foretrekke. Da vil *Spyder* eller *Visual Studio Code* fungere bedre. Du finner mer informasjon om de ulike editorene på bokens nettside.



Figur 1.3 I JupyterLab ser du lett hva som er kode, og hva som er resultatet

1.3 Vi starter med noen enkle utregninger

Vi kan bruke Python som en avansert kalkulator. Vi bruker * for multiplikasjon og / for divisjon. Potenser skrives slik: `4**3`. Det vil si at ** er det samme som «opphøyd i». Vi kan regne direkte i konsollen (vinduet til høyre i figur 1.2). Merk at i konsollen i figurene i denne boken blir det brukt `>>>` foran input i stedet for `In [1]:`.

```
1  >>> 1+2
2  3
3  >>> a = 4
4  >>> b = 5
5  >>> a*b
6  20
7  >>> a/b
8  0.8
9  >>> a**b
10 1024
11 >>> a = 10
12 >>> a
13 10
```

Legg merke til at vi kan lage variabler og gi dem verdier. I eksemplet ovenfor har vi definert to variabler, a og b og gitt dem verdiene $a = 4$ og $b = 5$ (linje 3 og 4). Da kan vi regne med disse videre. Vi kan også omdefinere dem, slik vi har gjort med a (som har fått den nye verdien 10, se linje 11). Tabell 1.1 viser ulike regneoperasjoner i Python.

Symbol	Operasjon	Eksempel	Resultat
+	Addisjon	1+2	3
-	Subtraksjon	23 - 11	12
*	Multiplikasjon	4.3*5.5	23.65
/	Divisjon	15/4	3.75
**	Potens	2**3	8
//	Heltallsdivisjon	10//4	2
%	Kongruens modulo	28 % 12	4

Tabell 1.1 Regneoperasjoner i Python

Merk at `%` gir oss restleddet ved divisjon av to tall:

```
1 >>> 21 // 8
2 2
3 >>> 21 % 8
4 5
```

Vi ser at `21 // 8` blir 2 siden $21/8 = 2{,}625$. Altså er heltallsdelen lik 2. Svaret på `21 % 8` blir 5 siden $21 = 2 \cdot 8 + 5$.

Oppgave 1.1

Åpne Spyder og prøv selv! Skriv inn regnestykkene i konsollen og trykk enter.

Oppgave 1.2

Bruk konsollen til å svare på følgende spørsmål:

a) Hva er $2 + 0{,}5 + 2^3 - \sqrt{12} + \dfrac{17}{2^5 - 1}$?

b) Hvor mange ganger går 8 opp i 100?

c) Er 39 812 904 031 delelig med 7?

d) Punktet $(1, 0)$ har rotert 45 321 grader om origo. Hvor mange hele runder har punktet da rotert? Hva er vinkelen mellom punktet og x-aksen?

Oppgave 1.3

Undersøk hvordan Python regnet ut `a % b` og `a // b` når

a) $a > 0$ og $b < 0$

b) $a < 0$ g $b > 0$

c) $a < 0$ og $b < 0$.

I konsollen vil vi kunne se svaret på et regnestykke når vi trykker enter. Dette gjelder ikke om vi skriver koden i en fil og kjører den (som vist i figur 1.2). Vi må be Python om å vise oss svaret ved å bruke print-funksjonen.

Eksempel 1.1

I kode (1.2) nedenfor vil vi få skrevet ut på skjermen (i konsollen) tallet 6 (som er $a + b$).

```
1 a=2
2 b=4
3 print(a+b)
```

Kode 1.2 Print-funksjonen sier hva Python skal skrive til skjermen ■

Vi kommer tilbake til hvordan vi kan formatere det vi printer (for eksempel antall desimaler og skrive tall på standardform).

1.4 Variabler i Python

Vi har allerede sett hvordan vi kan gi navn til ulike størrelser. I eksempel 1.1 gav vi tallet 2 navnet a, og tallet 4 fikk navnet b. Det er vanlig å kalle a og b for *variabler*, men strengt tatt er dette ikke to variable størrelser, slik vi bruker dette begrepet i matematikken. Bokstavene a og b refererer her til to gitte tall. Dette er noe ulikt måten vi vanligvis tenker om variable størrelser på, som når vi for eksempel skriver $f(x) = x^2 - 3$. Her kan verdiene til x variere. Vi vil likevel kalle slike navngitte tall for variabler i denne boken. I læreplanene for LK20 finner vi allerede etter 5. trinn mål om at elevene skal kunne bruke variabler når de programmerer.

Eksempel 1.2

Hvor mange Fahrenheit er 25 °C? Vi bruker da formelen

$$F = \frac{9}{5} \cdot C + 32$$

```
1 >>> C=25
2 >>> F=9/5*C+32
3 >>> print(F)
4 77.0
```

Vi ser på linje 4 at 25 °C er det samme som 77 °F. ■

Oppgave 1.4

Lag en kode som regner om 88 °F til Celsius.

I Python skilles det mellom tekst og tall. En tekst er et sett med tegn som for eksempel bokstaver og mellomrom som kommer etter hverandre. Vi kaller en slik «tekst» for en *string* (på norsk sier vi streng). Desimaltall kalles for *float*. Skal du definere en tekststreng, skriver vi teksten i "anførselstegn".

```
>>> fornavn="Ola"
>>> etternavn="Normann"
>>> print(fornavn , etternavn)
Ola Normann
```

Resultatet av koden ovenfor er Ola Norman. Legg merke til at vi skriver komma mellom hver av strengene i print-funksjonen.

Skriver vi inn `a="3"` vil variabelen `a` være en streng, ikke et tall. Derfor vil følgende kode gi en feilmelding i Python:

```
>>> a="3"
>>> b=4
>>> print(a+b)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Problemet her er at vi prøver å addere en streng til et heltall.

Python kan sette sammen to strenger. Strengen «sko» og strengen «le» blir strengen «skole» om vi setter sammen slik:

```
>>> a = "sko"
>>> b = "le"
>>> print(a+b)
skole
```

Når vi jobber med variabler i Python, er det lurt å bruke navn som er forklarende. I stedet for å kalle farten for x, kan vi like godt kalle den for «fart». Når du lager navn til variabler, er det et par ting du må være klar over:

– Navnet må starte med en bokstav eller understrekning (_). Resten kan være tall eller bokstaver.

– Navn er versalsensitive (case sensitive). Det vil si at det skilles mellom store og små bokstaver.

– Du kan ikke bruke såkalte Python-nøkkelord. Det vil si at du må unngå å bruke følgende navn:

  False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

Oppgave 1.5

Hva tror du følgende kode vil gi?

```
1 a="3"
2 b="4"
3 print(a+b)
```

Hva med følgende kode:

```
1 a=3
2 b=4
3 print(a+b)
```

Det er mange flere datatyper enn strenger og desimaltall i Python. Tabell 1.2 viser fem slike typer.

Kode	Datatype	Eksempel
`int`	heltall	`4343`
`float`	desimaltall	`3.14159`
`str`	streng (tekst)	`"Eksempeltekst"`
`bool`	boolsk verdi	`False`
`list`	liste	`[3, 4, "sko", -4]`

Tabell 1.2 Noen viktige datatyper i Python

Vi kan finne ut hvilken datatype et objekt har ved å bruke funksjonen `type()`: Vi kan konvertere en streng til for eksempel et heltall, dersom strengen består av tall:

```
>>> a = "4"
>>> type(a)
<class 'str'>
>>> b = int(a)
>>> type(b)
<class 'int'>
```

På linje 3 ser vi at a er en streng («str»). På linje 4 lager vi en ny variabel b som er a konvertert til et heltall.

1.5 Input fra bruker

Når vi lager program i Python, kan det være nyttig å be den som bruker programmet, om input av ulike størrelser. For å gjøre dette bruker vi funksjonen `input()`. Vi ser hvordan dette gjøres ved et par eksempler.

Eksempel 1.3

Lag et program som ber brukeren om lengden og bredden til et rektangel, og som deretter skriver ut hva arealet til rektanglet er.

Løsning:
Vi skriver følgende kode:

```
bredde = input("Skriv inn bredden av rektanglet: ")
lengde = input("Skriv inn lengden av rektanglet: ")
bredde = float(bredde) # Bredden er et desimaltall
lengde = float(lengde) # Lengden er et desimaltall
areal = lengde * bredde
print()
print("Arealet til rektanglet er ", areal)
```

Kode 1.3 Vi ber brukeren skrive inn bredde og lengde

Legg merke til at vi på linje 3 og 4 gjør om input til desimaltall (float). Hadde vi ikke gjort det, ville vi fått en feilmelding som sier at vi ikke kan multiplisere sammen to strenger.

Legg også merke til hvordan vi navnsetter de ulike variablene. Det er lurt å bruke navn som er selvforklarende. På linje 6 har vi skrevet inn en tom print-funksjon. Dette har vi kun gjort for å få et linjeskift. Dette gjør output litt penere.

Legg også merke til at vi har skrevet et par kommentarer. Det gjør vi ved å bruke # foran teksten. Da vil Python ignorere det som står bak #. Har du lengre kommentarer til programmet (som går over flere linjer), kan du gjøre dette ved å omslutte teksten med triple «anførselstegn» """, som vist nedenfor.

```
1  """
2  Dette er en tekst som kun er her for
3  å forklare den som leser programmet.
4  Den blir ignorert når programmet blir kjørt
5  """
6  v_0 = 8.2
7  a = 9.81
8  t = 3.2
9
10 s = v_0 * t+1/2 * a * t ** 2
11
12 print("Strekningen er", s, "meter")
```

Kode 1.4 Det er lurt å skrive kommentarer i programmet ■

Eksempel 1.4

Lag et program som spør etter radius til en sirkel, og som gir arealet som output.

Løsning:
For å kunne regne ut arealet må vi vite hva π er. Vi kunne selvsagt bare lage en variabel pi med størrelse 3,141592654. I stedet vil vi hente verdien for π fra math, som er en modul som følger med Python. På linje 1 nedenfor ser du hvordan vi gjør dette. Vi har hentet ut og brukt variabelen `pi` fra math. Denne har verdien 3,141592653589793 ≈ π.

```
1 from math import pi
2 r = float(input("Hva er radiusen?"))
3 areal = pi * r ** 2
4 print("Arealet er", areal)
```

Kode 1.5 Vi importerer π fra modulen math.
Du skal lære mer om import av moduler i kapittel 3.4

Kjører vi programmet og skriver inn at radiusen er 3, så får vi ut

```
Arealet er 28.274333882308138
```

Legg merke til at vi kan printe ut flere ting om gangen. På linje 4 har vi printet ut to objekter: en tekststreng ("Arealet er") og et tall (areal). Dette går fint, men vi er kanskje ikke helt fornøyd med antall desimaler i svaret. For å kunne styre dette må vi formatere det vi printer. ■

1.6 Formatering av print

Når vi bruker print-funksjonen, får vi som standard opptil 15 desimaler i svaret. Det kan derfor være fint å formatere det vi ønsker å skrive ut til skjermen. Det kan for eksempel være at vi ønsker å kun få med tre desimaler, eller at vi ønsker at svaret skal skrives på standardform. Før vi viser hvordan dette kan gjøres, vil vi komme med noen betraktninger knyttet til undervisning i programmering. Det er vår erfaring at det er lurt å vente med dette til elevene er blitt litt mer trygge i å programmere. Ellers blir det fort veldig mye å holde styr på og mange funksjoner som elevene må kunne.

Vi kan bruke format()-funksjonen til å formatere tallene vi printer. Syntaksen er

```
format(verdi, spesifisering av format)
```

Her spesifiseres formatet som «bredde.presisjon» etterfulgt av en bokstav som angir type.

Eksempel 1.5

```
1 >>> print(format(323.918, "9.2f"))
2    323.92
3 >>> print(format(323.918, ".4f"))
4 323.9180
```

I første linje har vi sagt at det skal være 9 karakterer bredt (inkludert desimalskilletegnet). Det er tre tomme plasser før tallet kommer, i tillegg til de seks plassene `323.92` tar. I linje 3 er det ikke presisert hvor mange karakterer tallet skal ha og hvor tallet skal plasseres (antall karakterer og mellomrom). Derfor står tallet helt til venstre i output på rad 4. I første linje har vi angitt 2 desimaler, mens i linje 4 er det angitt fire desimaler. Bokstaven `f` angir at output skal være et desimaltall (float). ■

Eksempel 1.6

Her er noen eksempler på formateringer:

```
1 >>> a = 0.03429
2 >>> # Standardform med 2 to desimalers nøyaktighet og
3 >>> # minst 10 tegn/mellomrom fra margen:
4 >>> print(format(a, "10.2E"))
5   3.43E-02
6 >>> # Standardform med 5 desimaler:
7 >>> print(format(a, ".5E"))
8 3.42900E-02
```

```
>>> # Tallet skrevet som prosent med to desimaler:
>>> print(format(a, ".2%"))
3.43%
>>> # Her skriver vi ut tallet 23 med 5 plaser fra venstre marg:
>>> print(format(23, "5d"))
   23
```

Bokstaven `E` angir at tallet skal skrives på standardform. I linje 10 har vi skrevet tallet som prosent med 2 desimalers nøyaktighet. I siste linje har vi printet tallet 23 slik at det er 3 tomme plasser før tallet. Det vil si at utskriften har 5 plasser totalt. Dette er nyttig når vi skal formatere tabeller.

Når vi vil printe ut kombinasjoner av tekst (strenger) og verdier av variabler, så kan f-strenger være nyttige. Dette er formaterte tekststrenger. Skal du bruke variabler eller utregninger i utskriften plasserer du disse omgitt av krøllparenteser.

Du bruker med andre ord f-strenger på følgende måte:

```
print(f"Tekst du vil skrive {variabel: formatering}")
```

De neste eksemplene viser hvordan dette fungerer.

Eksempel 1.7

```
>>> a = 3
>>> b = 34
>>> print(f"{a} + {b} = {a+b}")
3 + 34 = 37
```

Eksempel 1.8

Programmet nedenfor regner ut arealet til en sirkel med radius 3.

```
>>> from math import pi
>>> r = 3
>>> print(f"Arealet er {pi*r**2:.3f}")
Arealet er 28.274
```