



Programmering i matematikk

Knut Skrindo

Programmering i matematikk

Python for videregående skole

Knut Skrindo

1	Innledning	5
2	Brukergrensesnitt Spyder	5
2.1	Et første program	6
3	Grunnleggende programmering	6
3.1	Skrive til skjerm	6
3.2	Matematiske operatører	7
3.3	Konsollen	7
3.4	Variabler	8
3.5	Datatyper	12
3.6	Hente inn opplysninger fra brukeren	12
3.7	Sannhetsverdier og tester	13
3.8	Repeterende kode, løkker	16
3.9	Funksjoner	22
3.10	Lister	24
3.11	Feilsøking og hjelp	28
3.12	Kodestil	29
3.13	Oppgaver i grunnleggende programmering	29
4	Matematiske verktøy	31
4.1	Eksterne bibliotek	31
4.2	Grafer og diagrammer (plotting)	33
5	Matematiske metoder	38
5.1	Løse likninger: Halveringsmetoden	38
5.2	Simulere stokastiske forsøk	39
5.3	Numerisk derivasjon	41
5.4	Numerisk integrasjon: Trappesum	43

© Knut Skrindo 2022

3. utgave

Materialet er vernet etter åndsverkloven. Uten uttrykkelig samtykke er eksemplarfremstilling bare tillatt når det er hjemlet i lov eller avtale med Kopinor.

Dette heftet er laget til kurset «Programmering i matematikk i videregående skole». Hftet er produsert i samarbeid med Gyldendal Undervisning.

Konsulenter: Lektor Trond Simen Mundal Nettet, student Eina Bergem Jørgensen og student Anna Lina Petruseviciute Sjur. Tusen takk!

Digital versjon av heftet: <https://programmeringskurs.no/kurshefte.html>.

Henvendelser om heftet rettes til Knut Skrindo, knut@skrindo.no.

1 Innledning

Dette heftet er laget til kurset «Programmering i matematikk i videregående skole». Heftet inneholder forklaringer og eksempler på grunnleggende programmering, noen matematiske verktøy og et utvalg av matematiske metoder som er relevante for matematikkundervisning i videregående skole etter læreplanene i Fagfornyelsen.

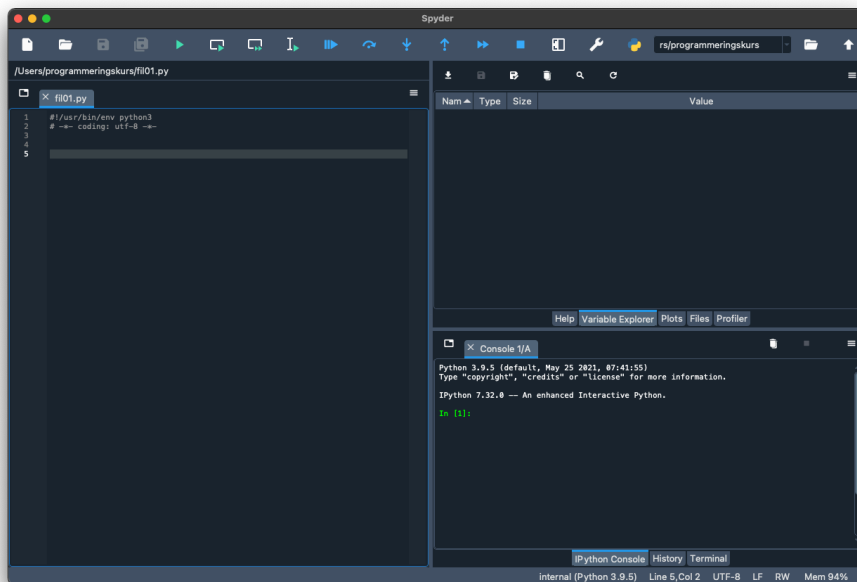
All programmering i heftet foregår i språket Python, versjon 3.

Løsningsforslag til oppgavene finner du på kursets nettside, <https://programmeringskurs.no/kurshefte.html>.

2 Brukergrensesnitt Spyder

Vi programmerer i Python med programmet Spyder. Andre alternativer er for eksempel Thonny, Jupyter, PyCharm, Atom eller Visual Studio Code. Python virker på samme måte i alle programmer, så det spiller liten rolle hvilket program du bruker. Vi installerer Spyder fra fra nettsiden til Spyder (<https://www.spyder-ide.org>) eller gjennom Anaconda (<https://anaconda.org>). En beskrivelse av hvordan du kommer i gang med Spyder finner du på kursets nettside, <https://programmeringskurs.no/komigang.html>.

Når du starter, ser Spyder-vinduet slik ut:



I feltet til venstre i vinduet ser du tekstfilen du programmerer i. I feltet øverst til høyre har du valget mellom «Help», «Variable explorer», «Plots» og «Files». Til vanlig kan det være nyttig å velge «Variable explorer», slik at du kan følge med

på hvilken verdi variablene dine har. Nederst til høyre har du Pythons konsoll. Her kan du skrive korte pythonuttrykk og se resultatet av dem med en gang. Dette er svært nyttig i testing og feilsøking.

Kommentar-tegnet i Python er «#». All tekst etter dette tegnet fram til neste linjeskift blir ignorert. Vi bruker kommentarer i koden for at den skal bli lettere å lese for andre.

2.1 Et første program

Vi lager en ny fil i Spyder og lagrer denne programfilen med et passende navn, for eksempel «fil01.py». For enkelhets skyld lar vi filnavnet slutte på «.py».

På første ledige linje skriver vi nå `print('Matematikk')`. Vi har skrevet vårt første program. Programmet er på én linje.

Vi skal nå kjøre programmet. I Spyder trykker vi på den grønne «Run file»-knappen. Nå skal teksten «Matematikk» komme til syne i konsollen.

3 Grunnleggende programmering

3.1 Skrive til skjerm

Når vi bruker kommandoen `print` sender vi tekst til konsollen. Vi kaller dette *å skrive til skjerm*. Det er vanlig å kalle teksten som kommer til syne i konsollen for programmets resultat eller output.

I `print`-kommandoen står tall og Python-uttrykk uten anførselstegn, mens tekst står med anførselstegn.

Vi lager et program med dette innholdet:

```
1 print('Regnestykket 3 + 5 er lik ')\n2 print(3+5)
```

Når vi kjører programmet, kommer denne teksten i konsollen:

```
Regnestykket 3 + 5 er lik\n8
```

Alt som står mellom anførselstegn blir gjengitt akkurat slik det står, mens det som står uten anførselstegn blir erstattet med verdien til uttrykket.

3.2 Matematiske operatører

Operator	Betydning	Eksempel	Resultat
+	addisjon	$3 + 4$	7
-	subtraksjon	$3 - 4$	-1
*	multiplikasjon	$3 * 4$	12
/	divisjon	$14 / 3$	4,6667
**	potens	$2 ** 3$	8
//	heltallsdivisjon	$14 // 3$	4
%	modulo	$14 \% 3$	2

De vanlige regneartene fungerer slik vi skulle tro. Vi bruker addisjon, subtraksjon, multiplikasjon og divisjon slik vi pleier på digitale verktøy. Det betyr at vi kan skrive $4 + 3$, $54 - 3$, $5 * 6$ og $24/3$, og resultatet blir som forventet.

Desimalkommaet er punktum. Potens skriver vi med to gangetegn: $2**3$ betyr 2^3 og blir 8.

EKSEMPEL 1

Regn ut $3 + 4 \cdot 5^3 - 4,53 + \frac{3^2-1}{2^3+1}$.

Løsning:

Vi skriver `print(3+4*5**3-4.53+(3**2-1)/(2**3+1))` og kjører programmet. I konsollen står det da 499.35888888888894.

Heltallsdivisjon er divisjon uten rest. Dette betyr at for eksempel både $24//8$ og $25//8$ blir 3, siden $24 : 8 = 3$ og $25 : 8 = 3 + \frac{1}{8}$.

Modulo-operatoren gir resten ved en divisjon. Så $5 \% 2$ gir 1, siden $5 : 2 = 2 + \frac{1}{2}$, altså 1 i rest. Videre får vi at $257 \% 11$ gir 4, siden $257 : 11$ har 4 i rest. For $257 : 11 = 23 + \frac{4}{11}$, altså $257 = 23 \cdot 11 + 4$.

Modulo-operatoren egner seg godt til å teste delelighet. Dersom et tall er delelig med et annet, så gir modulo-operatoren 0. Altså har vi at $24 \% 8$ gir 0, siden 24 er delelig med 8. Med vanlig matematisk notasjon skriver vi $24 \equiv 0 \pmod{8}$ og leser det som «tjuefire er ekvivalent med null modulo åtte».

3.3 Konsollen

Resultatet fra `print`-kommandoen kommer i konsollen. Konsollen kan også brukes til å finne verdien av små Python-uttrykk direkte, uten å bruke `print`.

EKSEMPEL 2

Regn ut $3 + 4.2 - 5 \cdot 11 + \frac{10.46}{2}$.

Løsning:

Vi skriver rett inn i konsollen og trykker enter.

In [1]: $3+4.2-5*11+10.46/2$

Out [1]: -42.569999999999999

OPPGAVER**1**

Skriv ut «Hello, World!» til skjerm.

2

Regn ut.

a) $3 + 4^2 - \frac{3}{2^4 + 1}$

b) $43,2 : 11$

3

a) Hvor mange ganger går 6 opp i 50?

b) Hva blir resten når du deler 50 på 6?

4

a) Avgjør om 167 335 er delelig med 3.

b) Avgjør om 64 er et partall.

5

Det er gått 115 469 timer siden klokka var midnatt.

Hva er klokka nå?

3.4 Variabler

3.4.1 Opprette en variabel

Du tilordner variabler verdi ved å skrive variabelens navn, etterfulgt av et likhetstegn og variabelens nye verdi. Tall skriver du rett fram, mens tekst må ha enkle eller doble anførselstegn rundt seg. Du kan velge om du vil bruke enkle eller doble tegn, bare du bruker samme tegn til å åpne og avslutte teksten.

EKSEMPEL 3

Her setter vi a til verdien 23, b får verdien «mønster» og c får verdien «matematikk».

1 $a = 23$

2 $b = \text{'mønster'}$

3 $c = \text{"matematikk"}$

Variabelnavn i python må oppfylle noen regler. Disse er:

- Variabler skal begynne med en vanlig bokstav.
- Variabler kan inneholde bokstaver, tall og understrek, `_`.
- Variabler kan ikke inneholde mellomrom eller noe annet.
- Noen ord er opptatt. Hvis du skriver inn et ord og det endrer farge, betyr det at ordet har spesiell betydning i programmet og at det derfor ikke kan brukes som variabelnavn.

Eksempler på akseptable variabler: `a`, `avvik`, `gjennomsnittlig_avvik`, `gjennomsnittligAvvik`, `avvik1`, `Avvik_2`.

Det kan være lurt å vende seg til å bruke variabelnavn som gir mening, for eksempel «gjennomsnitt» i stedet for «x». Da blir det lettere å lese koden.

3.4.2 Oppdatere en variabel

Likhetstegnet vi bruker til å tilordne variabler verdi må forstås som en tilordningsoperator, ikke som likhet. Når vi skriver `a = 3`, så blir verdien av `a` satt til 3, uansett hva verdien var før dette. Hvis vi derimot skriver `a = a + 4`, så blir verdien av `a` satt til summen av 4 og den verdien `a` hadde tidligere. Dette siste kan også skrives som `a += 4`. Denne hurtigskrivemåten virker med alle operatører.

EKSEMPEL 4

```
1 a = 3 # a har nå verdien 3
2 a = a + 4 # a har nå verdien 7
3 a += 5 # a har nå verdien 12
4 a *= 3 # a har nå verdien 36
```

Vi skriver variabelens verdi til skjerm med `print` uten anførselstegn.

EKSEMPEL 5

Skriv et program der du oppretter en variabel `m` med verdi 26. Lag deretter en ny variabel `n` som er det dobbelte av `m`. Skriv verdien av `n` til skjerm.

Løsning:

```
1 m = 26
2 n = 2*m
3 print(n)
```

Vi kjører programmet og får:

52

OPPGAVER

6

Skriv et program som gjør følgende:

- Opprett en variabel a med verdi 5.
- Legg til verdien 4.
- Legg til verdien 6.
- Legg til verdien 5.
- Skriv verdien av a til skjerm og kontroller at variabelen har verdien 20.

7Opprett to variabler a og b med verdiene 2 og 3. Bytt verdi, slik at a har b sin verdi og b har a sin verdi, uten å skrive inn tall på nytt.**8**Regn ut verdien av uttrykket $v_0t + \frac{1}{2}at^2$ med $v_0 = 5$, $a = 9,81$ og $t = 4$.**3.4.3 Utskrift av tekst og variabler sammen**

Når det vi skal skrive inneholder både tekst og verdien av variabler, bruker vi et såkalt f-literal. Vi bruker en «f» rett før anførselstegnet for å fortelle at det kommer et f-literal. Alt som står inni krøllparenteser blir tolket som pythonkode. Det betyr at $\{a\}$ blir tolket som verdien av variabelen a .

EKSEMPEL 6

```

1 a = 34/11
2 b = 1153
3 print(f"Her er a {a} og b {b}. Summen er {a + b}.")

```

Resultat:

```

Her er a 3.090909090909091 og b 1153. Summen er
1156.090909090909.

```

3.4.4 Avrunding og formatering i utskrift

F-literalene bruker vi også når vi vil runde av et tall i en printkommando. Rett før høyre mengdeparentes legger vi inn et kolon. Da kan vi styre hvordan utskriften formateres.

Vi runder av til tre desimaler med formatdefinisjonen «.3f». Her betyr bokstaven «f» at variabelen skal skrives ut med et fiksert antall desimaler og tallet betyr at det skal være tre desimaler. Hvis du i stedet skriver «.2f», får du to desimaler.

Avrundingen følger vanlige regler for avrunding.

EKSEMPEL 7

```

1 a = 34/11
2 b = 1153
3 print(f"Her er a {a:.2f} og b {b:.1f}. Summen er {a+b
   :.3f}.")

```

Resultat:

Her er a 3.09 og b 1153.0. Summen er 1156.091.

Hvis du skriver «.2e», så får du tallet på standardform, med to desimaler:

EKSEMPEL 8

```

1 a=3445/11
2 print(f"Tallet a er {a:.2e}")

```

Resultat:

Tallet a er 3.13e+02

OPPGAVER**9**

Skriv et program der du oppretter en variabel *tall* med verdien 56 og skriver ut verdien av variabelen med en svarsetning. Resultatet skal bli:

Verdien av variabelen tall er 56.

10

Skriv et program som legger verdien $\frac{11}{7}$ i en variabel. Skriv så ut verdien av variabelen med to desimaler.

11

Lagre tallet 1000 i en variabel. Bruk heltallsdivisjon og del variabelen på 23 og legg dette i en ny variabel. Resultatet av dette ganger du med 23 og legger i en tredje variabel. Skriv ut verdien av den siste variabelen.

12

Et rett prisme har sidekanter $l = 3$, $b = 4$ og $h = 5$. Skriv et program hvor du først setter verdien av l , b og h og deretter bruker variablene til å regne ut arealet av overflaten og volumet av prismet. Programmet skal

kun inneholde tallene 3, 4 og 5 én gang. Til slutt skal programmet skrive ut svarene, med en forlærende tekst.

3.5 Datatyper

De viktigste datatypene i Python er:

Kode	Datatype	Eksempel
str	streng (tekst)	'programmering'
int	heltall	1153
float	desimaltall (flyttall)	2.7182818
list	liste	['a', 3, 5, 1, 1, 1, 'b']
bool	boolsk verdi	True, False
set	mengde	{2, 3, 4}
np.array	np-array (datastruktur)	array([2, 1, 4, 3])

Noen av de matematiske operatorene har også betydning når de brukes med andre typer enn heltall og desimaltall. For eksempel betyr «+» sammensetning og «*» gjentakelse når de brukes på strenger og lister.

EKSEMPEL 9

Vi kan eksperimentere med operatorene + og * direkte i konsollen, det vil si vinduet nederst til høyre i Spyder.

```
In [1]: "fem" + "ti"
Out[1]: 'femti'
```

```
In [2]: "fem" * 3
Out[2]: 'femfemfem'
```

3.6 Hente inn opplysninger fra brukeren

Vi ber om informasjon fra brukeren med kommandoen «input()». For at du skal kunne bruke informasjonen til noe, tilordner du verdien til en variabel når du bruker «input()».

EKSEMPEL 10

```
1 svar = input("Skriv inn noe her: ")
2 print(f"Du skrev inn {svar}.")
```

Resultat:

```
Skriv inn noe her: Matematikk
Du skrev inn Matematikk.
```

Hvis du skal be brukeren om et tall, må du konvertere til riktig datatype, heltall (int) eller desimaltall (float). Det kan du gjøre direkte, samtidig med inputkommandoen. Alle kommandoer skal ha argumentet i parenteser, så pass på at du bruker dem systematisk.

EKSEMPEL 11

```
1 a = int(input("Skriv inn et tall her: "))
2 b = float(input("Skriv inn et tall til her: "))
3 print(f"Du skrev inn {a} og {b}.")
```

Resultat:

```
Skriv inn et tall her: 11
Skriv inn et tall til her: 32
Du skrev inn 11 og 32.0.
```

Her kan du se at 11 oppfattes som et heltall og 32 som et desimaltall, siden det står henholdsvis 11 og 32.0.

OPPGAVER

13

Be brukeren om først fornavn, deretter etternavn. Skriv ut en hilsen som inneholder fullt navn.

14

Be om et tall fra brukeren. Konverter tallet til et desimaltall. Legg svaret i en variabel. Skriv ut til skjerm hva brukeren tastet inn.

15

Be om et tall fra brukeren. Regn ut halvparten av tallet. Skriv svaret til skjerm.

16

Be om et tall fra brukeren. Regn ut hvor mange hele ganger det tallet er større enn 5. Skriv svaret til skjerm.

3.7 Sannhetsverdier og tester

3.7.1 Sannhetsverdier

Vi bygger opp utsagn med disse binære relasjonene:

Relasjon	Betydning
==	er lik
!=	er ikke lik
<	er mindre enn
<=	er mindre enn eller lik
>	er større enn
>=	er større enn eller lik

Python har to sannhetsverdier, True og False. Alle utsagn evalueres til sann eller usann og får derfor henholdsvis verdien True eller False.

EKSEMPEL 12

Vi prøver ut relasjonene i konsollen:

```
In [1]: 3 < 2  
Out[1]: False
```

```
In [2]: 3 > 2  
Out[2]: True
```

```
In [3]: 3 != 2  
Out[3]: True
```

```
In [4]: "matematikk" == "matematikk"  
Out[4]: True
```

Vi kan bygge opp større utsagn med konjugasjonene «not», «and» og «or»:

EKSEMPEL 13

Vi prøver ut konjugasjonene i konsollen:

```
In [1]: 3 < 2 and 3 > 2  
Out[1]: False
```

```
In [2]: 3 < 2 or 3 > 2  
Out[2]: True
```

```
In [3]: not(3 < 2)  
Out[3]: True
```

3.7.2 Vilkår (if-tester)

Vi lager en test i Python med kontrollordet «if», etterfulgt av et utsagn og et kolon. Så kommer kode som skal utføres hvis utsagnet evalueres til True. Dette

kan kombineres med kontrollordet «else», et kolon og til slutt en kode som skal utføres hvis utsagnet evalueres til False.

Kontrollordene `if`, `elif` og `else` skal ikke være innrykket, men resten av if-koden skal være innrykket ett nivå.

EKSEMPEL 14

```

1 n = 10
2 a = 30
3 if a > n:
4     print(f"{a} er større enn {n}")
5 else:
6     print(f"{a} er ikke større enn {n}")

```

Resultat:

30 er større enn 10

I eksempelet over blir output «30 er større enn 10», siden verdien av variabelen `a` er større enn verdien av variabelen `n`. Da blir utsagnet `a > n` tolket til True, og linje 4 blir utført.

Spyder sørger i utgangspunktet for riktig innrykk automatisk, men det kan være nødvendig å rette opp manuelt. Det lønner seg å lage innrykk med tabulator-tasten, til venstre for Q på tastaturet. Da blir det minst feil. Innrykkene må være like på hvert nivå.

Når du skal gjennomføre flere tester etter hverandre, kan du også bruke kontrollordet «elif», en forkortelse for «else if».

EKSEMPEL 15

```

1 tall = int(input("Skriv inn et tall: "))
2 if tall % 3 == 0:
3     print(f"{tall} er delelig med 3.")
4 elif tall % 3 == 1:
5     print(f"{tall} er ikke delelig med 3. Divisjonen
6     har 1 i rest.")
7 else:
8     print(f"{tall} er ikke delelig med 3. Divisjonen
9     har 2 i rest.")

```

Resultat:

```
Skriv inn et tall: 35
35 er ikke delelig med 3. Divisjonen har 2 i rest.
```

Det er ikke påkrevd å ha med «else». Det går fint å bruke en if-test uten «elif» eller «else».

EKSEMPEL 16

```
1 a = int(input("Skriv inn et heltall: "))
2 svar = "Variabelen a er større enn eller lik 5."
3
4 if a < 5:
5     svar = "Variabelen a er mindre enn 5."
6
7 print(svar)
```

Resultat:

```
Skriv inn et heltall: 34
Variabelen a er større enn eller lik 5.
```

OPPGAVER

17

Skriv et program som ber brukeren om et tall og så skriver en melding om tallet er positivt eller negativt.

18

Lag et program som ber brukeren om to tall og sjekker om tallene er like. Programmet skal skrive resultatet til skjerm.

19

Skriv et program som ber brukeren om et tall og så svarer om det er et oddetall eller partall.

20

Lag et program som ber brukeren om to tall og sjekker om tallene har forskjellig fortegn. Programmet skal skrive resultatet til skjerm.

3.8 Repeterende kode, løkker

Datamaskinens fortrinn kommer tydelig til syne når vi ser hvordan vi kan implementere kodegjentakelse. Vi skal se på to måter å programmere repeterende

kode, for-løkker og while-løkker.

3.8.1 While-løkke

Syntaksen for en while-løkke er kontrollordet «while», etterfulgt av en logisk test, altså et utsagn som blir evaluert til True eller False, og et kolon. Instruksjonene i while-løkka blir utført så lenge utsagnet blir evaluert til True. Alle instruksjoner i løkka skal rykkes inn ett nivå.

EKSEMPEL 17

```

1 n = 0
2 while n < 4:
3     print(n)
4     n = n + 1

```

Resultat:

```

0
1
2
3

```

I eksempelet over starter variabelen n på 0. For hver gang løkka utføres, skrives verdien av n ut, og så økes verdien av n med 1. Dette skjer helt til verdien av n ikke lenger er mindre enn 4.

Dette kan vi bruke til å lage verditabell for en funksjon:

```

1 x = 0
2 while x <= 2:
3     y = x**2 + 2*x - 1
4     print(f'{x} {y}')
5     x = x + .5

```

Koden ovenfor starter x på 0. Så lenge x er mindre enn eller lik 2 utføres følgende:

- y settes til $x^2 + 2x - 1$.
- x og y skrives til skjerm.
- Variabelen x økes med 0,5.

Når vi kjører koden, får vi:

```

0 -1
0.5 0.25

```

1.0 2.0
 1.5 4.25
 2.0 7.0

I programkoden for verditablellen ovenfor er de forskjellige tallverdiene angitt direkte i koden der de trengs. Koden blir ryddigere og derved lettere å lese og vedlikeholde om vi strukturerer den litt annerledes: Vi samler variabler og konstanter øverst i koden, og refererer til dem senere, se eksempelet nedenfor.

EKSEMPEL 18

Lag verditablell for $f(x) = x^2 + 2x - 1$ for $x \in [0, 2]$ med steglengde 0,5.

Løsning:

Vi bruker en `while`-løkke til å styre x -verdiene. Vi samler variabelene i starten av koden. Vi runder av verdiene når vi skriver dem ut.

```

1 xstart = 0 # Nedre grense for x
2 xslutt = 2 # Øvre grense for x
3 xsteg = 0.5 # Steglengden i verditablellen
4
5 x = xstart
6
7 while x <= xslutt:
8     y = x**2 + 2*x - 1
9     print(f'{x:.2f} {y:.2f}')
10    x = x + xsteg

```

While-løkker må ofte kombineres med en eller flere tellere eller midlertidige variabler. Vi tar med et eksempel hvor svaret på oppgaven er antall ganger løkka har kjørt.

EKSEMPEL 19

Lag et program som løser likningen $5^x = 625$ ved å telle hvor mange ganger 625 er delelig med 5.

Løsning:

Ideen her er at vi starter n på 625 og deler med 5 helt til det ikke går lenger. Og så teller vi underveis hvor lenge det går.

Vi gir variabelen n startverdi 625, se linje 1. Deretter oppretter vi en tellevariabel, se linje 2. Vi kjører `while`-løkka så lenge n er delelig med 5, altså har rest null om vi hadde delt på 5. Inni løkka deler vi n på 5 og lar ny n ha denne verdien. Og vi øker verdien av telleren med én.

```

1 n = 625

```

```

2 teller = 0
3 while n % 5 == 0:
4     teller += 1
5     n //= 5
6
7 print(teller)

```

Programmet gir 4 til svar, som betyr at $5^4 = 625$.

En feil i en while-løkke kan lett føre til en evig løkke. Da må du manuelt avbryte programmet ved å trykke på stopp-knappen i øvre del av konsollen, og så rette feilen.

OPPGAVER

21

Skriv et program som lager verditabell for $f(x) = -x^2 + 7x - 12$ med steglengde 0,1 for $x \in [-10, 10]$.

22

Skriv et program som skriver ut femgangen.

23

Skriv et program som skriver ut leddene i en tallfølge $\{a_i\}$ definert ved

$$a_1 = 4$$

$$a_n = 2 \cdot a_{n-1} + 1$$

Programmet skal slutte når a_i overstiger 1000.

Programmet skal altså skrive ut tallene 4, 9, 19, 39, 79, 159, 319 og 639.

24

Skriv et program som bruker en teller til å finne svaret i regnestykket $15 - 7$, uten å bruke operatoren minus.

Tips: Programmet skal bruke en løkke til å telle seg opp fra 7 til 15. Svaret skal vise at løkka har kjørt 8 ganger.

25

Skriv et program som løser likningen $2^n = 128$ ved å telle hvor mange ganger vi må opphøye 2 for å få 128.

26

Lag et program som skriver ut en hilsen, spør om brukeren vil se hilsenen en gang til, og fortsetter til brukeren svarer nei.

3.8.2 For-løkke

Når vi på forhånd ikke vet hvor mange ganger en løkke skal kjøre, må vi bruke `while`-løkke. Når vi derimot vet hvor mange ganger koden skal gjentas, bruker vi en for-løkke. Syntaksen er kontrollordet «for» etterfulgt av en fritt valgt variabel, kontrollordet «in» og til slutt et objekt som kan itereres over, for eksempel en mengde, og et kolon. Alle instruksjoner i løkka skal rykkes inn ett nivå.

Vi lager indeksmengder med kommandoen `range`. Kommandoen `range(3, 8)` betyr heltallene fra og med 3 til, men ikke med, 8.

EKSEMPEL 20

```
1 for i in range(3, 8):
2     print(i)
```

Resultat:

```
3
4
5
6
7
```

Hvis du utelater første argument i `range`, antar Python at første argument er 0. Dersom du legger til et tredje argument, angir dette steglengden i intervallet.

Kommando	Verdi
<code>range(0, 5)</code>	0, 1, 2, 3, 4
<code>range(5)</code>	0, 1, 2, 3, 4
<code>range(2, 5)</code>	2, 3, 4
<code>range(0, 5, 2)</code>	0, 2, 4
<code>range(10, 5, -1)</code>	10, 9, 8, 7, 6

Indeksen trenger ikke brukes inni løkka. Da brukes `range` til å bestemme hvor mange ganger løkka skal utføres.

EKSEMPEL 21

```
1 for i in range(3):
2     print("Matematikk")
```

Resultat:

```
Matematikk
```

Matematikk
Matematikk

Vi tar med et eksempel på en løkke som lager verditabell.

EKSEMPEL 22

Vi lager en verditabell for $f(x) = 3x^2 + 1$.

```
1 print("x  f(x)")
2 print("-----")
3 for x in range(0, 5, 2):
4     print(f"{x}  {3*x**2 + 1}")
```

Resultat:

x	f(x)
0	1
2	13
4	49

For-løkker kan også brukes med andre objekter enn de vi lager med `range`. Noen datatyper, som strenger og lister, kan man iterere over direkte.

EKSEMPEL 23

```
1 for bokstav in "søt":
2     print(bokstav)
```

Resultat:

s
ø
t

OPPGAVER

27

Skriv et program som skriver ut seksgangen.

28

Lag et program som skriver ut alle oddetallene mellom 0 og 100.

29

Lag et program som skriver ut de 100 første leddene i den geometriske følgen $\{3, 6, 12, 24, \dots\}$.

30

Skriv et program som gir denne outputen:

```
x
xxx
xxxxx
xxxxxxx
```

Tips: `'x'*4` gir `xxxx`.

3.9 Funksjoner

Funksjoner i programmering er en måte å samle kode i en sammensatt bolk, slik at det blir lettere å lese, skrive og vedlikeholde koden. Du kan klare deg uten funksjoner, men det blir lettere å programmere med funksjoner enn uten.

Syntaksen for funksjoner er kontrollordet «def», deretter navnet på funksjonen, to parenteser og et kolon. Dersom du vil at funksjonen skal kunne ta et eller flere argumenter, så skriver du dem inn i parentesene.

Koden nedenfor definerer $f(x) = x^2 + 5x - 12$:

```
1 def f(x):
2     return x**2 + 5*x - 12
```

Vi bruker funksjonen på vanlig måte, for eksempel $f(2)$, altså ved å skrive navnet, med eventuelle argumenter.

Hvis vi skal ta vare på verdien fra funksjonen, legger vi verdien inn i en variabel.

Vi kan klare oss uten funksjoner, men funksjoner gir ryddigere og enklere kode, som for eksempel i verditablellen nedenfor.

EKSEMPEL 24

Lag verditablell til $f(x) = x^3 - x^2$ for $x \in [0, 1]$ med steglengde 0,2.

Løsning:

```
1 def f(x):
2     return x**3 - x**2
3
4 xstart = 0
5 xslutt = 1
6 xsteg = .2
```

```

7
8 x = xstart
9 y = f(x)
10
11 while x <= xslutt:
12     print(f'{x:.3f} {y:.3f}')
13     x = x + xsteg
14     y = f(x)

```

Når vi kjører koden får vi:

```

0.000 0.000
0.200 -0.032
0.400 -0.096
0.600 -0.144
0.800 -0.128
1.000 0.000

```

Funksjoner kan ha flere argumenter. Da skiller vi argumentene med komma i definisjonen.

EKSEMPEL 25

Vi skal skrive en funksjon som legger sammen to tall.

```

1 def addisjon(n, m):
2     return n + m
3
4 summen = addisjon(5, 7)
5 print(summen)

```

Resultat:

```
12
```

Funksjoner kan også defineres uten noe `return`-utsagn. Når du bruker en slik funksjon, blir alle funksjonens linjer med kode utført, men funksjonen returnerer ingen verdi.

OPPGAVER

31

Lag en funksjon f som returnerer det samme som den matematiske funksjonen $f(x) = 3x^2 + 3$. Bruk deretter f til å skrive ut $f(0)$, $f(-3)$ og $f(5)$.

32

Skriv et program som lager verditabell for $f(x) = x^2 - 7x + 1$ med steglengde 0,1 for $x \in [-5, 5]$. Bruk f som en funksjon underveis i programmet.

33

Lag en funksjon som returnerer produktet av to tall.

34

Lag en funksjon som adderer to tall og skriver ut regnestykket og svaret.

3.10 Lister**3.10.1 Å lage lister selv**

Lister er elementer adskilt med komma mellom hakeparenteser, for eksempel `[1, 2, 3]` og `['Maria', 'Siri', 7]`.

Du oppretter en tom liste ved å tilordne verdien `[]` til en variabel, for eksempel `liste=[]`.

Når vi skal legge til elementer i en liste, bruker vi kommandoen «append». Denne bruker vi med listevariabelen som forstavelse, adskilt med et punktum.

EKSEMPEL 26

```
1 liste = []
2 liste.append(5)
3 liste.append(2)
4 liste.append('a')
5 print(liste)
```

Resultat:

```
[5, 2, 'a']
```

Når vi skal bygge opp en tallfølge i en liste bruker vi løkker til å legge tallene inn i lista, se linje 15 og 16 i eksempelet nedenfor.

EKSEMPEL 27

La f være funksjonen gitt ved

$$f(x) = -0,4x^2 - 2$$

Skriv et program som legger x -verdiene og y -verdiene fra verditabellen til f inn i to lister. Velg x mellom -2 og 2 med steglengde $0,5$.

Løsning:

```
1 def f(x):
2     return -.4*x**2 - 2
3
4 xstart = -2
5 xslutt = 2
6 xsteg = .5
7
8 xverdier = []
9 yverdier = []
10
11 x = xstart
12 y = f(x)
13
14 while x <= xslutt:
15     xverdier.append(x)
16     yverdier.append(y)
17     x = x + xsteg
18     y = f(x)
19
20 print(xverdier)
21 print(yverdier)
```

Koden ovenfor gir følgende resultat:

```
[-2, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0]
[-3.6, -2.9, -2.4, -2.1, -2.0, -2.1, -2.4, -2.9, -3.6]
```

Alle kommandoer vi bruker på lister angis med listevariabelen som forstavelse.

3.10.2 Hente ut elementer fra en liste

Lister er itererbare objekter, så vi kan løpe gjennom dem med en `for`-løkke.

EKSEMPEL 28

Vi løper gjennom en liste med en `for`-løkke.

```
1 liste = [1, 2, 3, 4]
2 for tall in liste:
3     print(tall)
```

Når vi kjører programmet får vi:

```
1
```

2
3
4

Vi plukker ut bestemte elementer fra en liste med hakeparenteser. `liste[0]` gir første element, `liste[1]` gir andre element, `liste[2]` gir tredje element og så videre. `liste[-1]` gir siste element.

For å plukke ut en lengre del av listen, bruker vi intervallnotasjon med kolon, for eksempel `liste[2:4]`. Vi teller fra 0, så dette betyr tredje og fjerde, men ikke femte element.

EKSEMPEL 29

Vi tar utgangspunkt i lista definert som

```
liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Kommando	Resultat	Forklaring
<code>liste[0]</code>	<code>'a'</code>	Første element er nummer 0.
<code>liste[0:3]</code>	<code>['a', 'b', 'c']</code>	Tar ikke med nummer 3 (fjerde).
<code>liste[:3]</code>	<code>['a', 'b', 'c']</code>	Fra starten til nr. 3.
<code>liste[-1]</code>	<code>'h'</code>	Siste element er nummer -1.
<code>liste[3:6]</code>	<code>['d', 'e', 'f']</code>	Fra nummer 3 til nummer 6.
<code>liste[3:-1]</code>	<code>['d', 'e', 'f', 'g']</code>	Tar ikke med siste element.
<code>liste[3:]</code>	<code>['d', 'e', 'f', 'g', 'h']</code>	Tar med siste element.

Vi tar også med kontrollordet `len`, som gir oss lengden av listen, altså antall elementer i listen.

EKSEMPEL 30

```
1 liste=[6,2,5,3,5,6,6]
2 print(len(liste))
```

Resultat:

7

Vi kan bruke `len`-kommandoen sammen med en løkke til å gjentatte ganger

velge ut bestemte elementer i lista. Kommandoen `len(range())` lager en indeksemengde med like mange elementer som lista har.

EKSEMPEL 31

Regn ut differansen mellom de seks første kvadrattallene.

Løsning: `len(liste)` gir antall elementer i lista, mens `range(len(liste))` gir en indeksemengde med samme lengde. Antall differanser er én mindre enn antall elementer i lista, så `for`-løkka har indeksemengde `range(len(liste) - 1)`.

```
1 liste = [1, 4, 9, 16, 25, 36]
2
3 for i in range(len(liste) - 1):
4     differanse = liste[i + 1] - liste[i]
5     print(differanse)
```

Vi kjører programmet og får:

```
3
5
7
9
11
```

OPPGAVER**35**

Opprett en liste som består av tallene fra 0–9.

36

Opprett en tom liste. Legg til tre navn i listen.

37

Opprett en tom liste. Legg til noen elementer. Skriv ut hvor mange elementer det er i listen.

38

Opprett en liste med 10 elementer. Plukk ut tredje, fjerde og femte element.

39

En liste er gitt som

```
liste=[1, 4, 9, 16, 25, 36]
```

Bruk en `for`-løkke til å regne ut produktet av alle elementene i lista.

40

Lag et program som lager en tom liste, ber brukeren om et tall og så bruker en `while`-løkke til å legge til ettall i lista helt til lengden av lista er lik tallet brukeren la inn.

41

Skriv et program som definerer funksjonen $f(x) = x^2 + 3x - 1$. Programmet skal be brukeren om én og én x -verdi og legge disse inn i en liste. Når brukeren avslutter, skal programmet regne ut de tilhørende funksjonsverdiene og legge dem inn i en egen liste. Til slutt skal de to listene skrives til skjerm.

3.11 Feilsøking og hjelp

Alle som skriver programmer opplever feil. Det er derfor nyttig å venne seg til å lese feilmeldinger. I feilmeldingen får du beskjed om hvilken linje feilen er på, en beskrivelse av feilen og hvilken type det er. Med litt trening, får du god hjelp i programmeringen av feilmeldingene.

De vanligste feilene er disse:

- `SyntaxError`: Du har ikke overholdt rettskrivningsreglene. Har du glemt kolon? Har du noen ekstra tegn? Har du brukt parenteser feil? Er det feil ved innrykk? Ofte er feilen på linja over den feilmeldingen sier at feilen er på.
- `IndexError`: Vanligvis betyr dette at du prøver å plukke ut et element fra en liste, og så har du gjort en tellefeil i starten eller slutten av lista. Sjekk nøye hvor mange elementer det er i lista og hvilke elementer du skal plukke ut.
- `NameError`: Feil som ofte gjelder variabler. Kanskje du har stavet variabelen feil? Eller glemt å opprette variabelen før du bruker den?
- `TypeError`: Du bruker feil datatype i en funksjon, for eksempel bruker et desimaltall som argument i `range`.

Når du feilsøker, kan det være god hjelp å se hvilken verdi variablene dine har underveis når programmet kjører. Dette får du til ved å legge inn printkommandoer på utvalgte steder. I tillegg kan du se på «Variable Explorer» i Spyder, som viser verdien av variablene etter at programmet er ferdig.

Når du er litt usikker på hva koden din egentlig gjør, kan du teste deler av den i konsollen, slik vi gjør i eksemplene på side 14 og 26.

Hvis du ikke finner feilen, kommer du ofte langt med å google «python» + feilmeldingen.

Du finner mye god dokumentasjon av Python på nettet. Et fint sted å starte å se etter hjelp er The Python Tutorial, <https://docs.python.org/3/tutorial/>. Hvis det gjelder plotting, kan du lese <https://matplotlib.org>, som er matplotlib sine sider. Hvis det gjelder det numpy, kan se på <https://www.numpy.org/>.

3.12 Kodestil

Etter hvert som vi begynner å beherske de grunnleggende elementene i programmering, kan det være på sin plass å bruke litt tid på kodens utseende. Da kan du lese PEP8 (<https://www.python.org/dev/peps/pep-0008/>), den offisielle anbefalingen for hvordan Pythonkode bør se ut. Følger du anbefalingene i PEP8, så øker lesbarheten av koden. Som lærere kommer vi til å lese mye kode, så det er en fordel for oss om både kolleger og elever skriver lettlest kode.

3.13 Oppgaver i grunnleggende programmering

Du har nå vært gjennom grunnleggende programmering. Nå kan du øve på mange ulike problemstillinger i programmering. Her er et utvalg oppgaver.

OPPGAVER

42

Skriv et program som legger partallene mellom 0 og 100 inn i en liste.

43

- Lag en funksjon «gjennomsnitt(*a*, *b*)», som returnerer gjennomsnittet av *a* og *b*.
- Lag et program som for hvert av partallene mellom 1 og 10 skriver ut gjennomsnittet av tallet og det påfølgende oddetallet.

Det du får i konsollen skal være:

```
2.5
4.5
6.5
8.5
```

44

La *f* være funksjonen gitt ved

$$f(x) = x^2 + 2x + 3$$

- Lag en verditabell for *f*(*x*). Velg selv grenser og steglengde.
- Utvid programmet ditt til å la brukeren velge minste og største verdi for *x* og trinnene, i tabellen.

45

Skriv et program som ber brukeren om et heltall *n* og så summerer tallene $\{1, 2, 3, \dots, n\}$.

46

Skriv et program som ber brukeren om et oddetall n , sjekker at n er odde og summerer oddetallene $\{1, 3, 5, \dots, n\}$.

47

Lag et program som ber om et heltall n og skriver ut alle tall i 6-gangen som er mindre enn n .

48

Lag en funksjon som bruker operatoren «*» og en løkke til å regne ut x^n , uten å bruke «**».

49

Skriv et program som ber brukeren om et tall og så skriver til skjerm hvor mange sifre tallet har.

50

Lag et program som skriver ut de 100 første Fibonaccitallene.

51

Skriv et program som gir denne outputen:

```
x
xxx
xxxxx
xxxxxxx
```

52

Skriv et program som lager en liste med primtallene mellom 0 og 100.

53

Lag et program som ber bruker om et tall og så finner tallets største primtallsfaktor.

54

Be om et heltall n og finn neste perfekte tall, altså minste tall større enn n som er slik at tallet er summen av alle divisorer som er mindre enn n .

4 Matematiske verktøy

4.1 Eksterne bibliotek

Python består av en grunnpakke med kommandoer som virker uten videre. Når vi vil bruke andre kommandoer, laster vi inn tilleggspakker fra såkalte biblioteker. Disse bibliotekene er spesielt nyttige for oss:

- `matplotlib`: Brukes til å tegne grafer.
- `math`: Brukes til matematiske operasjoner.
- `random`: Brukes til å generere tilfeldige tall.
- `pandas`: Brukes til å laste inn eksterne datasett.
- `numpy`: Brukes til å lage og regne med såkalte arrayer, en type vektorer.

Eksterne biblioteker kan lastes inn på flere måter. Den anbefalte måten å gjøre det på er å bruke kommandoen «`import`», etterfulgt av biblioteket og eventuelt et alias. Denne metoden gjør det lett å se hvor kommandoene er hentet fra. For `pyplot`, `pandas` og `numpy` er konvensjonen å bruke alias «`plt`», «`pd`» og «`np`».

```
1 import matplotlib.pyplot as plt
2 import math
3 import random
4 import pandas as pd
5 import numpy as np
```

Kommandoene i bibliotekene blir nå tilgjengelige med biblioteknavnet som forstavelse, for eksempel «`math.pi`» for π , «`plt.plot(x, y)`» for å tegne en graf, og «`random.randint(1,10)`» for å trekke et tilfeldig heltall mellom 1 og 10.

`Numpy` introduserer en egen datatype, en såkalt `numpy-array`. Vi bruker disse blant annet til å lagre tallverdier, for eksempel når vi skal tegne kurver.

Vi kan opprette `numpy-array` på flere måter. Kommandoen `arange` virker på omtrent samme måte som `range`, men den er ikke begrenset til heltall.

EKSEMPEL 32

```
1 import numpy as np
2 verdier = np.arange(3,3.5,.1)
3 print(verdier)
```

Resultat:

```
[3.  3.1 3.2 3.3 3.4]
```

Her er variabelen «`verdier`» nå av typen `numpy-array`. Endepunktet er normalt ikke inkludert, slik det heller ikke er når vi bruker `range`.

For oss har en numpy-array to viktige fordeler:

- Vi kan bruke desimaltall i steglengden.
- Vi kan regne ledd for ledd, omtrent som vektorer.

EKSEMPEL 33

```
1 import numpy as np
2 x = np.arange(1,5,.5)
3 y = x**2
4 print(x)
5 print(y)
```

Resultat:

```
[1.  1.5 2.  2.5 3.  3.5 4.  4.5]
[ 1.   2.25  4.   6.25  9.  12.25 16.  20.25]
```

OPPGAVER

55

Importer biblioteket `math`. Test kommandoene «`math.pi`» og «`math.e`». (Undersøk også hva som skjer uten forstavelsen «`math.`».)

56

Importer biblioteket `random`, og test kommandoen «`random.randint(1,6)`» og «`random.random()`» noen ganger. Hva gjør de?

57

La f være funksjonen $f(x) = x^2 + 3x + 1$. Bruk en numpy-array til å lage en verditabell for f . Bestem selv hvilke x -verdier du vil ha i verditabellen.

58

Lag et program som løser andregradslikninger med løsningsformelen. Programmet skal be om koeffisientene a , b og c , teste om diskriminanten er negativ, null eller positiv og skrive ut løsningen(e) til skjerm.

59

Skriv et program som lar brukeren spille stein–saks–papir mot datamaskinen flere ganger. Etter hver runde skal stillingen vises.

Noen foretrekker å importere slik: `from math import *`. Da lastes alle kommandoer fra biblioteket inn og kan brukes uten forstavelse, altså `pi` i stedet for `math.pi`. Det er fordeler og ulemper med begge metodene, men den anbefalte metoden er å importere slik vi har beskrevet ovenfor.

4.2 Grafer og diagrammer (plotting)

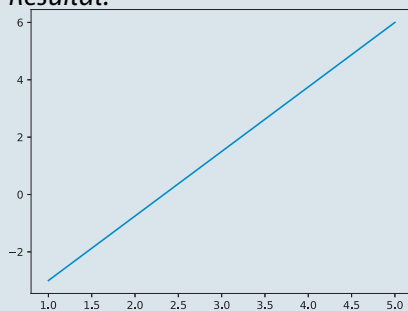
Vi tegner grafer med det eksterne biblioteket `pyplot` fra `matplotlib`. Vi importerer det med `import matplotlib.pyplot as plt`. Alle kommandoer i `pyplot` blir nå tilgjengelige med forstavelsen «`plt`».

Den viktigste kommandoen er «`plot`». Vanligvis plotter vi ved å gi `plot` to lister, som tilsvarer henholdsvis x -verdiene og y -verdiene fra en verditabell.

EKSEMPEL 34

```
1 import matplotlib.pyplot as plt
2 plt.plot([1, 5], [-3, 6])
```

Resultat:

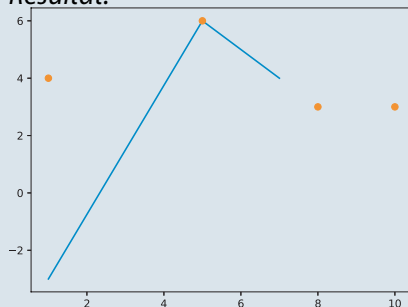


Initialverdien til `plot` er å trekke en linje gjennom alle punktene. Dersom vi legger til et tredje argument, `'o'`, får vi i stedet tegnet opp punktene.

EKSEMPEL 35

```
1 import matplotlib.pyplot as plt
2 plt.plot([1, 5, 7], [-3, 6, 4])
3 plt.plot([1, 5, 8, 10], [4, 6, 3, 3], 'o')
```

Resultat:



Python kan ikke tegne glatte kurver, bare rette streker mellom punkter. For å tegne kurver som ser glatte ut, må vi derfor plote mange punkter. Det er vanlig

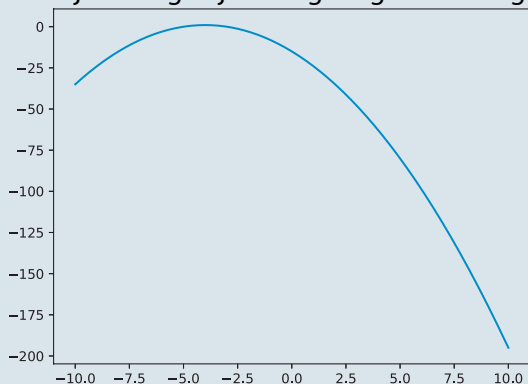
å bruke en numpy-array, se side 31, til dette.

EKSEMPEL 36

Vi skal tegne grafen til $f(x) = -x^2 - 8x - 15$ for $x \in [-10, 10]$.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 xstart = -10
5 xslutt = 10
6 steg = .1
7 x = np.arange(xstart, xslutt, steg)
8
9 def f(x):
10     return -x**2 - 8*x - 15
11
12 plt.plot(x, f(x))
```

I linjene 1 og 2 laster vi inn numpy og matplotlib. I linjene 4–5 setter vi noen initialverdier for x og oppretter variabelen x . Linjene 9–10 definerer funksjonen og linje 12 tegner grafen. Det gir denne grafen:

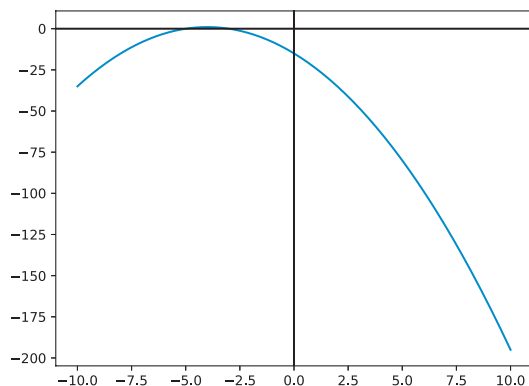


4.2.1 Endre grafens utseende

Vi kan endre grafens utseende, slik at vi får den slik vi vil. Det første vi savner er x -akse og y -akse. Disse kan vi tegne med `plt.axhline` og `plt.axvline`.

```
13 plt.axhline(color='black')
14 plt.axvline(color='black')
```

Hvis vi legger disse to linjene til i eksempelet over får vi:



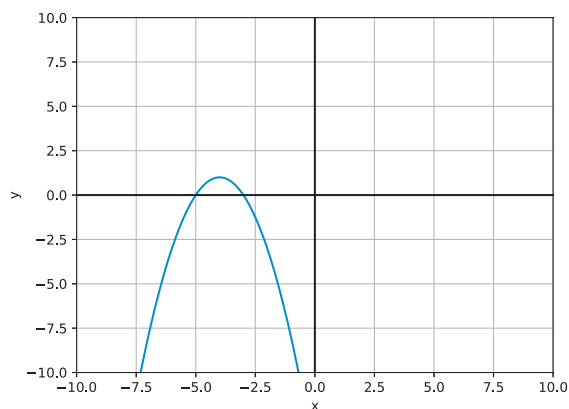
Utsnittet i vinduet blir ikke alltid slik vi ønsker det automatisk. Hvis du vil stille det inn manuelt, kan du bruke `xlim` og `ylim`.

```
15 plt.xlim(-10,10)
16 plt.ylim(-10,10)
```

Aksenavn får du med `xlabel` og `ylabel`. Rutenettet slår du på med `grid()`.

```
17 plt.xlabel('x')
18 plt.ylabel('y')
19 plt.grid()
```

Den samlede koden ovenfor gir denne grafen:



Det er mulig å få grafen til å se ut omtrent akkurat hvordan du vil. Oversikt over kommandoer, innføringer og eksempler finner du på nettsiden til `matplotlib`, <https://matplotlib.org>.

Nedenfor ser du et eksempel på kode som tegner to grafer omtrent slik vi er vant til å tegne for hånd:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
```

```
4 # Setter øvre og nedre grense for x og y.
5 xstart = -10
6 xslutt = 10
7 ystart = -10
8 yslutt = 10
9
10 # Definerer steglengden, altså hvor tett punktene er.
11 steg = .1
12 # Oppretter numpy-array med x-verdier.
13 x = np.arange(xstart, xslutt, steg)
14
15 # Definerer funksjonene vi skal tegne grafen til.
16 def f(x):
17     return -x**2 - 8*x - 15
18 def g(x):
19     return (1/3)*x - 4
20
21 # Plotter funksjonene.
22 plt.plot(x, f(x), label="f(x)")
23 plt.plot(x, g(x), label = "g(x)")
24
25 # Viser grafforklaringen/etiketten.
26 plt.legend()
27
28 # Setter øvre og nedre grenser for hvilke x og y som vises:
29 plt.xlim(xstart, xslutt)
30 plt.ylim(ystart, yslutt)
31
32 # Setter navn på aksene.
33 plt.xlabel('x')
34 plt.ylabel('y', rotation = 0)
35
36 # Viser rutenett.
37 plt.grid()
38
39 # Flytter aksene slik at de går gjennom origo.
40 # Lagrer aksene i en variabel.
41 akser = plt.gca()
42 # Flytter nedre og venstre akse slik at de går gjennom origo
43
44 akser.spines['bottom'].set_position('zero')
45 akser.spines['left'].set_position('zero')
46 # Skjuler øvre og høyre akse.
47 akser.spines['top'].set_visible(False)
```

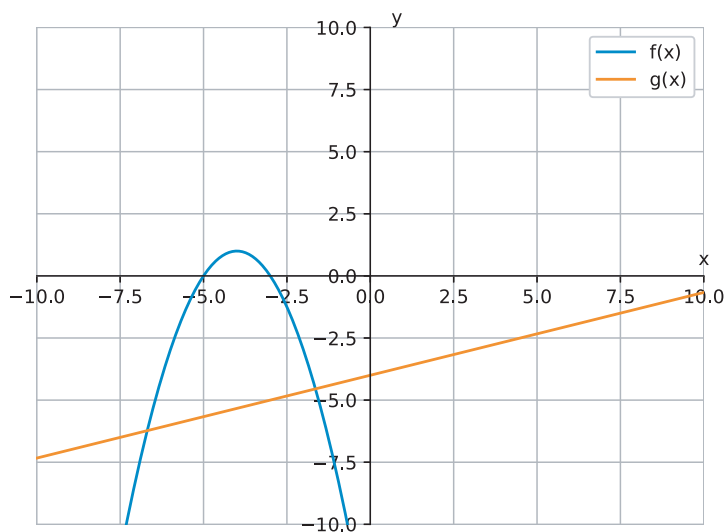
```

47 akser.spines['right'].set_visible(False)
48
49 # Flytter aksenavnene så de synes.
50 akser.yaxis.set_label_coords(0.54,1)
51 akser.xaxis.set_label_coords(1,0.55)

```

I linje 50 og 51 er koordinatene et tall mellom 0 og 1. Koordinatene vi har brukt her passer i dette tilfellet, men vil ikke alltid stemme med grafen.

Koden gir denne grafen:



I linje 13 har vi opprettet `x` med `np.arange`, hvor vi har definert ønsket steglengde. En annen tilnærming kunne være å i stedet definere hvor mange punkter vi ønsker. Da bruker vi `linspace` fra `numpy`. Linje 13 kunne da være `x = np.linspace(xstart, xslutt, 1000)`.

Det er ikke vanskelig å finne andre dataprogrammer som er lettere å bruke til å tegne grafer. Koden ovenfor er med for å demonstrere noen muligheter.

OPPGAVER

60

Tegn punktene i et koordinatsystem: $\{(0, 30), (5, 43), (10, 55), (15, 60)\}$.

61

Tegn grafen til $f(x) = x^2 + 3x - 4$ for $x \in [-5, 5]$.

5 Matematiske metoder

Nedenfor finner du eksempler på hvordan vi kan løse ulike matematiske problemer numerisk.

5.1 Løse likninger: Halveringsmetoden

Enhver likning $f(x) = g(x)$ kan skrives om til $f(x) - g(x) = 0$. Vi setter $h(x) = f(x) - g(x)$, og får at likningen har en løsning når $h(x) = 0$. Så å løse likninger er det samme som å finne nullpunkter til funksjoner. Det finnes flere metoder å gjøre dette på. En variant er halveringsmetoden.

Halveringsmetoden tar utgangspunkt i et intervall hvor funksjonsverdien i endepunktene til intervallet ligger på hver sin side av x -aksen. Hvis funksjonen har et nullpunkt, så må det finnes et slikt intervall. Halveringsmetoden går ut på at vi halverer intervallet. Da må nullpunktet ligge i en av halvdelene. Vi velger den halvdelene der endepunktene har forskjellig fortegn. Vi gjentar metoden med dette nye intervallet, helt til endepunktene ligger så nærme hverandre vi ønsker. Endepunktene er da en tilnærming for nullpunktet.

EKSEMPEL 37

Vi skal løse likningen $3x - 4 = 0$. Vi legger merke til at med $f(x) = 3x - 4$, så har $f(10)$ og $f(-10)$ forskjellige fortegn. Vi bruker derfor -10 og 10 som grenser.

```
1 def f(x):
2     return 3*x - 4
3
4 # Initialverdier
5 nedregrense = -10
6 øvregrense = 10
7 presisjon = .01
8
9 # Setter opp variablene a, b og m.
10 a = nedregrense
11 b = øvregrense
12 m = (a + b)/2
13
14 while b - a > presisjon:
15     if f(a) * f(m) < 0:
16         b = m
17     elif f(b) * f(m) < 0:
18         a = m
19     elif f(m) == 0:
20         break
21     else:
```

```

22     print(f"f({a}) og f({b}) ligger på samme side
      av x-aksen.")
23     break
24     m = (a + b)/2
25
26 print(f"Funksjonen har et nullpunkt i x = {m:.3f}.")

```

Funksjonen har et nullpunkt i $x = 1.333$.

Halveringsmetoden finner bare én løsning, uavhengig av om det eksisterer flere løsninger.

OPPGAVER

62

Skriv et program som løser likningen med halveringsmetoden.

- $3x - 4 = \frac{3}{2}x - 7$
- $x^2 + 3x - 5 = 0$ (Finn begge løsningene!)
- $x + \sin x = 4$ for $x \in [0, 10]$ (Tips: bruk `math.sin()`.)
- $x \ln x = 2$ (Tips: bruk `math.log()`.)

63

Utvid programmet fra oppgave 62 til å telle antall halveringer.

64

Halveringsmetoden deler intervallet $[a, b]$ i to. Skriv et program som løser likningen ved å dele intervallet i tre deler.

5.2 Simulere stokastiske forsøk

Vi kan simulere stokastiske forsøk ved å la Python generere tilfeldige tall. Til dette importerer vi biblioteket «random». De mest aktuelle funksjonene fra dette biblioteket er `random.randint`, som trekker ut et tilfeldig heltall, `random.choice`, som trekker ut et tilfeldig objekt fra en liste, og `random.shuffle`, som stokker en liste.

EKSEMPEL 38

Vi simulerer terningkast med `random.randint(1, 6)`. Dette programmet gjennomfører 10 terningkast og skriver ut antall øyne:

```

1 import random
2 n = 10
3 terninger = []
4 for i in range(n):

```

```
5     terninger.append(random.randint(1,6))
6     print(terninger)
```

Et mulig resultat kan bli dette:

```
[6, 2, 1, 6, 4, 2, 2, 5, 3, 2]
```

EKSEMPEL 39

Vi velger fra en liste med funksjonen `random.choice()`. Dette programmet lager en plan for de neste 10 valgene dine i stein-saks-papir:

```
1     import random
2     n = 10
3     valg = ['stein', 'saks', 'papir']
4     plan = []
5     for i in range(n):
6         plan.append(random.choice(valg))
7     print(plan)
```

Et mulig resultat er dette:

```
['papir', 'papir', 'papir', 'papir', 'stein', 'papir',
 'stein', 'saks', 'stein', 'saks']
```

OPPGAVER

65

Skriv et program som simulerer et myntkast 1000 ganger og teller opp hvor mange mynt og hvor mange kron det ble.

66

Skriv et program som simulerer et terningkast 100 ganger og teller opp hvor mange ganger hver av mulighetene forekommer.

67

Skriv et program som trekker ut tre elementer fra en liste på ti elementer, uten å legge tilbake etter hvert trekk.

5.3 Numerisk derivasjon

Vi tar utgangspunkt i definisjonen til den deriverte av en funksjon $f(x)$:

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

For små verdier av Δx kan vi derfor finne en tilnæringsverdi for $f'(x)$ slik:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

EKSEMPEL 40

Vi lar f være funksjonen $f(x) = x^2 + 3x$. Vi skal regne ut en tilnæringsverdi for $f'(4)$ med $\Delta x = 0,001$.

```

1 def f(x):
2     return x**2 + 3*x
3
4 def derivert(a, deltax):
5     return (f(a + deltax) - f(a))/deltax
6
7 a = 4
8 deltax = .001
9
10 fder = derivert(a, deltax)
11 print(fder)

```

Resultat:

11.001000000003813

Vi tar med et eksempel til hvor vi tegner grafen til den deriverte.

EKSEMPEL 41

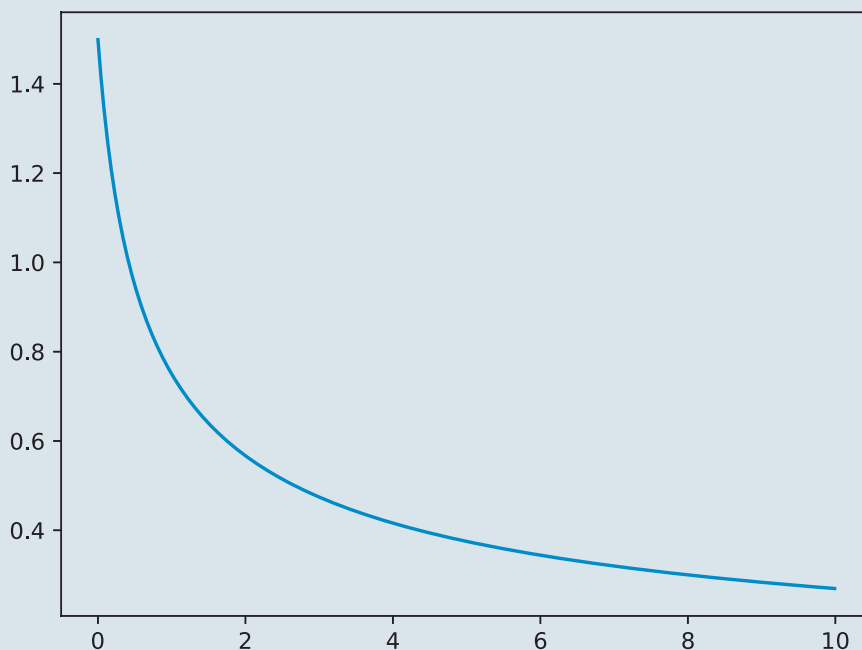
Vi tegner grafen til den deriverte til $f(x) = \sqrt{3x + 1}$ mellom 0 og 10.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x):
5     return np.sqrt(3*x + 1)
6
7 def derivert(a, deltax):
8     return (f(a + deltax) - f(a))/deltax
9

```

```
10 xstart = 0
11 xslutt = 10
12 deltax = .001
13 antall_punkter = 1000
14
15 x = np.linspace(xstart, xslutt, antall_punkter)
16 y = derivert(x, deltax)
17 plt.plot(x, y)
```

**OPPGAVER****68**La f være funksjonen gitt ved

$$f(x) = \sqrt{x^2 + 3x}$$

- Skriv et program som bestemmer en tilnæringsverdi for $f'(3)$.
- Skriv et program som tegner grafen til $f(x)$ og $f'(x)$ i samme koordinatsystem.

5.4 Numerisk integrasjon: Trappesum

Det bestemte integralet av en funksjon $f(x)$ fra a til b kan vi litt forenklet definere slik:

$$\int_a^b f(x) dx = \lim_{n \rightarrow \infty} \sum_a^b f(x_i) \cdot \Delta x$$

Dette betyr at vi deler opp intervallet $[a, b]$ i n deler, og så summerer vi alle rektangler med bredde $\Delta x = \frac{b-a}{n}$ og høyde $f(x_i)$, der x_i er x -verdiene som utgjør intervallenes grenser i intervallet a og b .

Vi kan finne en numerisk tilnærming for det bestemte integralet på flere måter. En metode er å dele opp intervallet mange ganger, og så summere rektanglene. Altså finner vi trappesummen.

EKSEMPEL 42

Vi finner en numerisk tilnærming av $\int_2^5 \sqrt{x} dx$ med venstresidig trappesum.

```

1 import numpy as np
2 # Setter opp innstillinger
3 xstart = 2
4 xslutt = 5
5 antall_partisjoner = 100000
6 delta_x = (xslutt - xstart)/antall_partisjoner
7 # Hvilken funksjon skal integreres
8 def f(x):
9     return np.sqrt(x)
10 # Lager liste over venstre x-verdi til alle
    partisjonene (med bredde delta_x)
11 xverdier = np.linspace(xstart, xslutt, antall_partisjoner
    )
12 integral = 0
13 for x in xverdier[:-1]:
14     integral += f(x)*delta_x
15 print(f"{integral:.3f}")

```

5.568

OPPGAVER

69

Bestem en numerisk tilnærming for integralene med venstresidig trappesum.

a) $\int_1^3 \sin x \, dx$

b) $\int_0^5 e^{2x} \, dx$

c) $\int_{-2}^5 x^2 + 3 \, dx$

70

I metoden ovenfor brukte vi en venstresidig trappesum. Dersom funksjonen var voksende, ble dette den nedre trappesummen. Bestem en numerisk tilnærming for integralene i oppgave 69 med høyresidig trappesum.

71

Bestem en numerisk tilnærming for integralene i oppgave 69 der du summerer rektangler hvor høyden er funksjonsverdien av midtpunktet i hvert intervall.

72

Bestem en numerisk tilnærming for integralene i oppgave 69 ved å summere trapeser i stedet for rektangler.

Videre læring

Første kursdag i kurset «Programmering i matematikk i videregående skole» har fokusert på grunnleggende programmering i Python og sett på noen eksempler på matematiske verktøy og metoder. Hvordan skal dere jobbe i tiden etter kurset for å bli bedre forberedt til å undervise i programmering i matematikktimene?

Programmeringsverksted: Det kan være lurt å prøve å arrangere noen programmeringsverksteder. Bli enige på forhånd hvilke oppgaver dere skal se på. Lag løsningsforslag på egenhånd og sammenlikne løsninger i plenum.

Dagskurs 2: Python i klasserommet: Du lærer flere matematiske metoder og lærer om hvordan vi underviser i programmering i matematikk.

- Vilkår og løkker i flere sammenhenger.
- Eksterne biblioteker.
- Arbeid med funksjoner og grafer.
- Numerisk derivasjon.
- Numerisk løsning av likninger.
- Utforskende undervisning med programmering.
- Programmeringsdidaktikk.

Dagskurs 3: Python i S og R, simulering og modellering: Du lærer hvordan vi kan simulere stokastiske forsøk i Python og hvordan vi kan arbeide med modellering i Python.

- Modellering av datasett, blant annet med lineære, eksponentielle og logistiske modeller.
- Regresjon i Python.
- Simulering av stokastiske forsøk.

<https://programmeringskurs.no/>

Register

- addisjon, 7
- arange, 31
- avrunding, 10

- bibliotek, 31

- datatyper, 12
- derivasjon, 41
- divisjon, 7

- eksterne bibliotek, 31

- f-literal, 10
- False, 14
- feilsøking, 28
- for-løkke, 20
- formatering, 10
- funksjon, 22

- halveringsmetoden, 38
- heltallsdivisjon, 7

- if-test, 14
- IndexError, 28
- integrasjon, 43

- konsoll, 6
- kontrollstruktur, 14

- linspace, 37, 42
- liste, 24
- løkke, 16

- math, 31
- matplotlib, 31

- modulo, 7
- multiplikasjon, 7

- NameError, 28
- np.arange, 31
- numerisk derivasjon, 41
- numerisk integrasjon, 43
- numerisk løsning av likning, 38
- numpy, 31
- numpy-array, 31

- operatorer, 7

- pandas, 31
- plot, 33
- potens, 7

- random, 31
- relasjoner, 13

- skrive til skjerm, 10
- Spyder, 5
- standardform, 11
- subtraksjon, 7
- SyntaxError, 28

- tegne graf, 33
- trappesum, 43
- True, 14
- TypeError, 28
- typer, 12

- variabel, 8

- while-løkke, 17

Kommandooversikt

Kommando	Forklaring	Sidetall	Kommando	Forklaring	Sidetall
+	addisjon	7	math.e	$e \approx 2,718281828$	31
-	subtraksjon	7	math.log	lg	39
*	multiplikasjon	7	math.pi	$\pi \approx 3,141592654$	31
/	divisjon	7	math.sin	sinus	39
**	potens	7	np.arange	lage liste	31
//	heltallsdivisjon	7	np.linspace	array med jevnt fordelte tall	37
%	modulo (rest)	7	np.zeros	array med 0-tall	
<	mindre enn	14	or	eller	14
<=	mindre enn eller lik	14	plt.axhline	x-akse	34
>	større enn	14	plt.axvline	y-akse	34
>=	større enn eller lik	14	plt.gca	'get current axis'	35
==	lik	14	plt.grid	rutenett	35
!=	ikke lik	14	plt.legend	grafforklaring	35
and	og	14	plt.plot	tegne graf	33
append	legge til element (liste)	24	plt.xlabel	navn på x-aksen	35
def	definisjon (funksjon)	22	plt.xlim	x-intervall	35
elif	ellers hvis (if)	15	plt.ylabel	navn på y-aksen	35
else	ellers (if)	15	plt.ylim	y-intervall	35
False	usann	14	print	skrive til skjerm	6
float	gjøre om til flyttall/desimaltall	13	random.choice	tilfeldig utvalg fra liste med tilbakelegging	39
for	for (løkke)	20	random.randint	tilfeldig heltall	39
from	fra (import)	32	random.shuffle	stokke liste	39
if	hvis	15	range	tellemengde	20
import	importere	31	return	hva funksjonen skal returnere	23
input	hente data fra brukeren	12	True	sann	14
int	gjøre om til heltall	13	while	mens (løkke)	17
len	lengde (streng, liste)	26			
not	ikke	14			

