Programmering i matematikk

i ungdomsskolen

Knut Skrindo

Programmering i matematikk i ungdomsskolen

Python for ungdomsskolen

Knut Skrindo

1. utgave

Dette heftet er laget til kurset «Programmering i matematikk i ungdomsskolen». Heftet er produsert i samarbeid med Gyldendal Undervisning.

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

Digital versjon av heftet: https://programmeringskurs.no/.

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

1 Innledning

Dette heftet er laget til kurset «Programmering i matematikk i ungdomsskolen». Heftet inneholder forklaringer og eksempler på grunnleggende programmering, samt noen matematiske verktøy, som er relevante for matematikkundervisning i ungdomsskolen etter læreplanene i Fagfornyelsen.

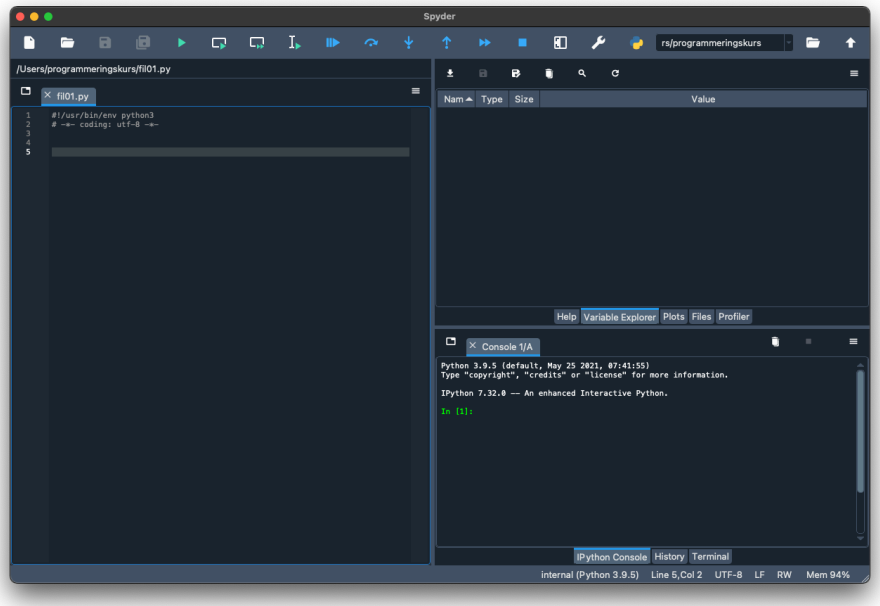
All programmering i heftet foregår i språket Python, versjon 3.

Løsningsforslag til oppgavene finner du på kursets nettside, https://programmeringskurs.no/.

2 Brukergrensesnitt Spyder

Vi programmerer i Python med programmet Spyder. Andre alternativer er for eksempel Thonny, Mu, Jupyter, PyCharm, Atom eller Visual Studio Code. Du kan også programmere i en nettleser, for eksempel i Trinket eller repl.it. 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/usk/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 ')
2 print(3+5)
```

Når vi kjører programmet, kommer denne teksten i konsollen:

```
Regnestykket 3 + 5 er lik
8
```

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 operatorer

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
%	rest	`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

a) `4 + 3`

b) `54 - 3`

c) `5 * 6`

d) `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 $25 : 8 = 3 + \frac{1}{3}$.

Rest-operatoren gir resten ved en divisjon. Så `5 % 2` gir 1, siden $5 : 2$ har 1 i rest. Tilsvarende har vi at `257 % 11` gir 4, siden $257 : 11$ har 4 i rest.

Rest-operatoren egner seg godt til å teste delelighet. Dersom et tall er delelig med et annet, så gir rest-operatoren 0. Altså har vi at `24 % 8` gir 0, siden 24 er delelig med 8.

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]: -42.56999999999999
```

OPPGAVER

1
Skriv ut «Hello, World!» til skjerm.

2
Regn ut.

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

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 = 'mønster'
3 c = "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 operatorer.

EKSEMPEL 4

```
a = 3 # a har nå verdien 3
a = a + 4 # a har nå verdien 7
a += 5 # a har nå verdien 12
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:

```
m = 26
n = 2*m
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 4.
- Legg til 6.
- Legg til 5.
- Skriv verdien av *a* til skjerm og kontroller at variabelen har verdien 20.

7
Opprett en variabel. Lag en annen variabel som er det dobbelte av den første variabelen.

8
Opprett 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.

9
Regn ut verdien av uttrykket $v_0 t + \frac{1}{2} a t^2$ med $v_0 = 5$, $a = 9{,}81$ og $t = 4$.
Tips: Det er vanlig å skrive «v_0» som `v0` eller `v_0`.

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 tretallet 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

10
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.
```

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

12
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.

13
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 forlarende 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 operatorerne + 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 input-kommandoen. 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

14
Be om et tall fra brukeren. Konverter tallet til et desimaltall. 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.

17
Be brukeren om først fornavn, deretter etternavn. Skriv ut en hilsen som inneholder fullt navn.

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

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

Resultat:

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

Det går fint å bruke en if-test uten «elif» eller «else».

EKSEMPEL 16

```
a = int(input("Skriv inn et heltall: "))
svar = "Tallet er større enn eller lik 5."

if a < 5:
    svar = "Tallet er mindre enn 5."

print(svar)
```

Resultat:

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

OPPGAVER

18
Skriv et program som ber brukeren om et tall og så skriver en melding om tallet er positivt eller negativt.

19
Lag et program som ber brukeren om to tall og sjekker om tallene er like. Programmet skal skrive resultatet til skjerm.

20
Skriv et program som ber brukeren om et tall og så svarer om det er et oddetall eller partall.

21
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

```
n = 0
while n < 4:
    print(n)
    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:

```
x = 0
while x <= 2:
    y = x**2 + 2*x -1
    print(f'{x} {y}')
    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 verditabellen 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 anerledes: Vi samler variabler og konstanter øverst i koden, og refererer til dem senere, se eksempelet nedenfor.

EKSEMPEL 18

Lag verditabell 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 verditabellen
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 telle-variabel, se linje 2. Vi kjører while-løkka så lenge n er delelig med 5, altså har rest mnull 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

22
Skriv et program som lager verditabell for $f(x) = -x^2 + 7x - 12$ med steglengde 0,1 for $x \in [-10, 10]$.

23
Skriv et program som skriver ut femgangen.

24
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.

25
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.

26
Skriv et program som løser likningen $2^n = 128$ ved å telle hvor mange ganger vi må opphøye 2 for å få 128.

27
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, 89` betyr heltallene fra og med 3 til, men ikke med, 9.

EKSEMPEL 20

```
1 for i in range(3, 9):
2     print(i)
```

Resultat:

```
3
4
5
6
7
8
```

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
`range(0, 5)`	0, 1, 2, 3, 4
`range(5)`	0, 1, 2, 3, 4
`range(2, 5)`	2, 3, 4
`range(0, 5, 2)`	0, 2, 4
`range(10, 5, -1)`	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

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

Resultat:

```
Matematikk
Matematikk
Matematikk
```

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

EKSEMPEL 22

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

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

Resultat:

```
x  y
-------
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

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

Resultat:

```
s
ø
t
```

OPPGAVER

28
Skriv et program som skriver ut seksgangen.

29
Lag et program som skriver ut alle oddetallene mellom 0 og 100.

30
Lag et program som skriver ut de 100 første leddene i den geometriske følgen $\{3, 6, 12, 24, \ldots\}$.

31
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 verditabellen nedenfor.

EKSEMPEL 24

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

Løsning:

```
def f(x):
    return x**3 - x**2

xstart = 0
xslutt = 1
xsteg = .2

x = xstart
y = f(x)

while x <= xslutt:
    print(f'{x:.3f} {y:.3f}')
    x = x + xsteg
    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.

```
def addisjon(n, m):
    return n + m

summen = addisjon(5, 7)
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

32
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 $f(5)$.

33
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.

34
Lag en funksjon som returnerer produktet av to tall.

35
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:

```
def f(x):
    return -.4*x**2 - 2

xstart = -2
xslutt = 2
xsteg = .5

xverdier = []
yverdier = []

x = xstart
y = f(x)

while x <= xslutt:
    xverdier.append(x)
    yverdier.append(y)
    x = x + xsteg
    y = f(x)

print(xverdier)
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.

```
liste = [1, 2, 3, 4]
for tall in liste:
    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
`liste[0]`	`'a'`	Første element er nummer 0.
`liste[0:3]`	`['a', 'b', 'c']`	Tar ikke med nummer 3 (fjerde).
`liste[:3]`	`['a', 'b', 'c']`	Fra starten til nr. 3.
`liste[-1]`	`'h'`	Siste element er nummer -1.
`liste[3:6]`	`['d', 'e', 'f']`	Fra nummer 3 til nummer 6.
`liste[3:-1]`	`['d', 'e', 'f', 'g']`	Tar ikke med siste element.
`liste[3:]`	`['d', 'e', 'f', 'g', 'h']`	Tar med siste element.

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

EKSEMPEL 30

```
liste=[6,2,5,3,5,6,6]
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 indeksmengde 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 indeksmengde med samme lengde. Antall differanser er én mindre enn antall elementer i lista, så `for`-løkka har indeksmengde `range(len(liste) - 1)`.

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

for i in range(len(liste) - 1):
    differanse = liste[i + 1] - liste[i]
    print(differanse)
```

Vi kjører programmet og får:

```
3
5
7
9
11
```

OPPGAVER

36
Opprett en liste som består av tallene fra 0–9.

37
Opprett en tom liste. Legg til tre navn i listen.

38
Opprett en tom liste. Legg til noen elementer. Skriv ut hvor mange elementer det er i listen.

39
Opprett en liste med 10 elementer. Plukk ut tredje, fjerde og femte element.

40
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.

41
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.

42
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 print-kommandoer 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.

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

43

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

44

a) Lag en funksjon «gjennomsnitt(a, b)», som returnerer gjennomsnittet av *a* og *b*.
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
```

45
La f være funksjonen gitt ved

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

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

46
Skriv et program som ber brukeren om et heltall n og så summerer tallene $\{1, 2, 3, \ldots, n\}$.

47
Skriv et program som ber brukeren om et oddetall n, sjekker at n er odde og summerer oddetallene $\{1, 3, 5, \ldots, n\}$.

48
Lag et program som ber om et heltall n og skriver ut alle tall i 6-gangen som er mindre enn n.

49
Lag en funksjon som bruker operatoren «*» og en løkke til å regne ut x^n, uten å bruke «**».

50
Skriv et program som ber brukeren om et tall og så skriver til skjerm hvor mange sifre tallet har.

51
Lag et program som skriver ut de 100 første Fibonaccitallene.

52
Skriv et program som gir denne outputen:

```
   x
  xxx
 xxxxx
xxxxxxx
```

53
Skriv et program som lager en liste med primtallene mellom 0 og 100.

54
Lag et program som ber bruker om et tall og så finner tallets største primtallsfaktor.

55
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.

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 og pandas er konvensjonen å bruke alias «plt» og «pd».

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

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.

OPPGAVER

56
Importer biblioteket math. Test kommandoene «math.pi» og «math.e». (Undersøk også hva som skjer uten forstavelsen «math.».)

57
Importer biblioteket random, og test kommandoen «random.randint(1,6)» og «random.random()» noen ganger. Hva gjør de?

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)

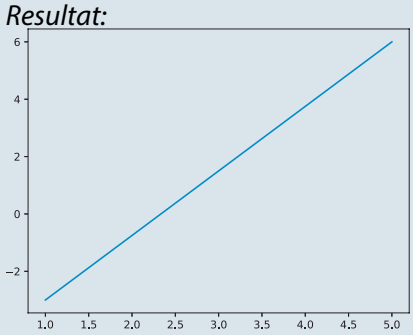
Vi tegner grafer med det eksterne biblioteket `pyplot` fra matplotlib. Vi importerer det med `import matplotlib.pyplot as plt`. Alle kommandoer i pyplot blir nå tilgjengelige med forstavelsen «plt.».

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 32

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

Resultat:

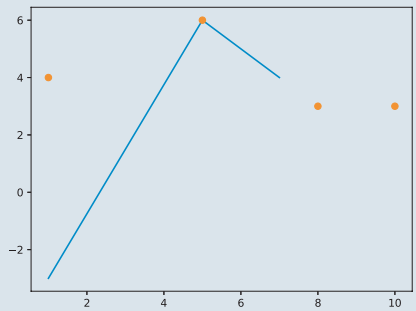


Initialverdien til plot er å trekke en linje gjennom alle punktene. Dersom vi legger til et tredje argument, `'o'`, får vi i stedet tegnet opp punktene.

EKSEMPEL 33

```
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 plotte mange punkter.

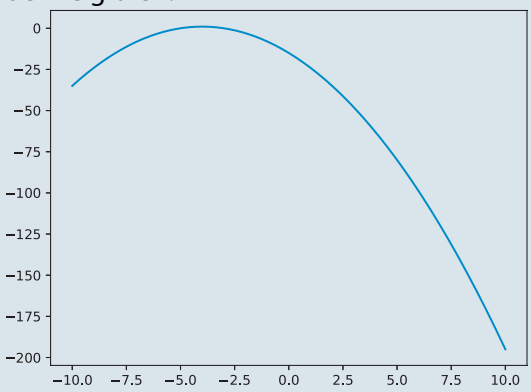
EKSEMPEL 34

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

```
1  import matplotlib.pyplot as plt
2
3  def f(x):
4      return -x**2 - 8*x - 15
5
6  # Setter øvre og nedre grense for x og y.
7  xstart = -10
8  xslutt = 10
9  xsteg = .1
10 # Oppretter lister til x og y
11 xverdier = []
12 yverdier = []
13
14 # Fyller listene med x- og y-verdier
15 x = xstart
16 y = f(x)
17
18 while x <= xslutt:
19     xverdier.append(x)
20     yverdier.append(y)
21     x = x + xsteg
22     y = f(x)
23
24 # Plotter funksjonene.
25 plt.plot(xverdier, yverdier)
```

I linjene 1 laster vi inn matplotlib. Linjene 3–4 definerer funksjonen. I linjene 6–12 setter vi noen initialverdier for x og oppretter lister. I linjene

14–22 fyller vi listene med *x*- og *y*-verdier. Linje 25 tegner grafen. Det gir denne grafen:

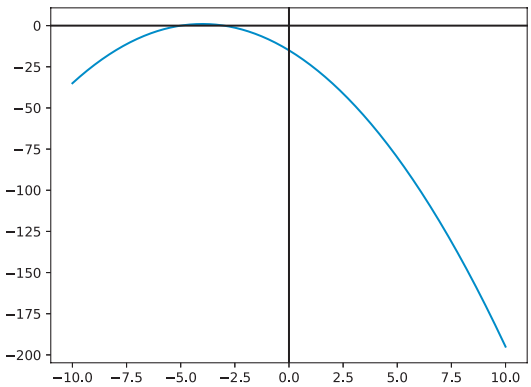


4.2.1 Endre grafens utseende

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`.

```
26 plt.axhline(color='black')
27 plt.axvline(color='black')
```

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



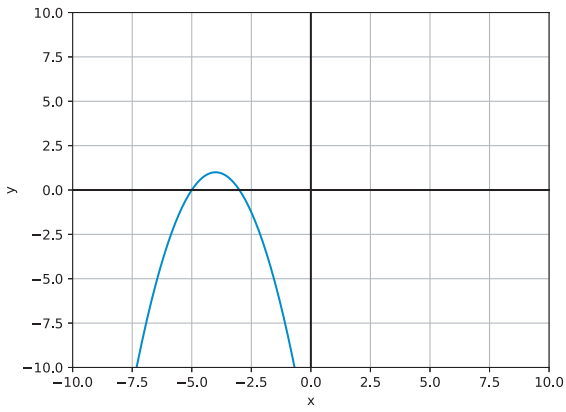
Utsnittet i vinduet blir ikke alltid slik vi ønsker det automatisk. Hvis du vil stille det inn manuelt, kan du bruke `xlim` og `ylim`.

```
28 plt.xlim(-10,10)
29 plt.ylim(-10,10)
```

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

```
30 plt.xlabel('x')
31 plt.ylabel('y')
32 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.

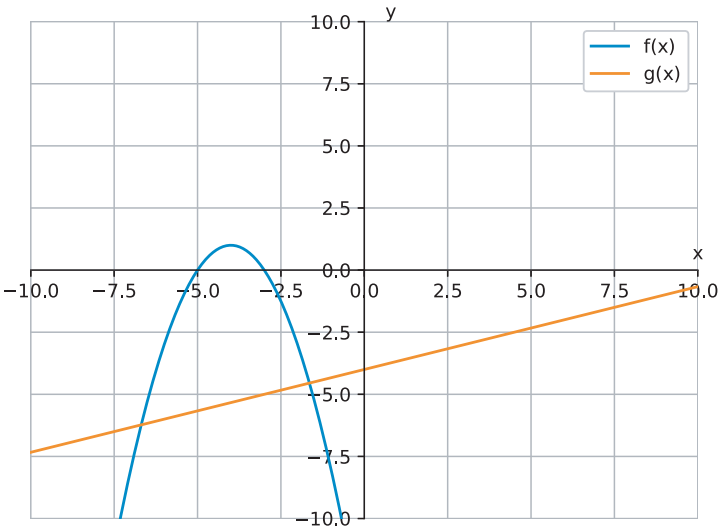
Nedenfor ser du et eksempel på kode som tegner to grafer omtrent slik vi er vant til å tegne for hånd:

```
import matplotlib.pyplot as plt

# Definerer funksjonene vi skal tegne grafen til.
def f(x):
    return -x**2 - 8*x - 15
def g(x):
    return (1/3)*x - 4

# Setter øvre og nedre grense for x og y.
xstart = -10
xslutt = 10
ystart = -10
yslutt = 10

# Definerer steglengden, altså hvor tett punktene er.
xsteg = .1
# Oppretter lister til x og y
xverdier = []
yverdier1 = []
yverdier2 = []

# Fyller listene med x- og y-verdier
x = xstart
y1 = f(x)
y2 = g(x)
```

```
26
27 while x <= xslutt:
28     xverdier.append(x)
29     yverdier1.append(y1)
30     yverdier2.append(y2)
31     x = x + xsteg
32     y1 = f(x)
33     y2 = g(x)
34
35 # Plotter funksjonene.
36 plt.plot(xverdier, yverdier1, label="f(x)")
37 plt.plot(xverdier, yverdier2, label = "g(x)")
38
39 # Viser grafforklaringen/etiketten.
40 plt.legend()
41
42 # Setter øvre og nedre grenser for hvilke x og y som vises:
43 plt.xlim(xstart, xslutt)
44 plt.ylim(ystart, yslutt)
45
46 # Setter navn på aksene.
47 plt.xlabel('x')
48 plt.ylabel('y', rotation = 0)
49
50 # Viser rutenett.
51 plt.grid()
52
53 # Flytter aksene slik at de går gjennom origo.
54 # Lagrer aksene i en variabel.
55 akser = plt.gca()
56 # Flytter nedre og venstre akse slik at de går gjennom origo
       .
57 akser.spines['bottom'].set_position('zero')
58 akser.spines['left'].set_position('zero')
59 # Skjuler øvre og høyre akse.
60 akser.spines['top'].set_visible(False)
61 akser.spines['right'].set_visible(False)
62
63 # Flytter aksenavnene så de synes.
64 akser.yaxis.set_label_coords(0.54,1)
65 akser.xaxis.set_label_coords(1,0.55)
```

I linje 64 og 65 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:



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 $x \in [-5, 5\rangle$.

4.3 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 35

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 36

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']
```

EKSEMPEL 37

Lag et program som simulerer 10 000 kast med to terninger. Be brukeren gjette summen av de to terningene, og skriv ut andelen som stemmer.

```
1  import random
2
3  N = 10000
4  gjett = int(input('Skriv inn hva du gjetter at summen
       blir: '))
5
6  vinst = 0
7  for i in range(N):
8      terning1 = random.randint(1, 6)
9      terning2 = random.randint(1, 6)
10     resultat = terning1 + terning2
11     if gjett == resultat:
12         vinst += 1
13
14 print(vinst/N)
```

OPPGAVER

62
Skriv et program som simulerer et myntkast 1000 ganger og teller opp hvor mange mynt og hvor mange kron det ble.

63
Skriv et program som simulerer et terningkast 100 ganger og teller opp hvor mange ganger hver av mulighetene forekommer.

64
Skriv et program som trekker ut tre elementer fra en liste på ti elementer, uten å legge tilbake etter hvert trekk.

65
Nina spiller på et lykkehjul. Lykkehjulet er nummerert fra 1 til 100. Hvert tall forekommer fire ganger på hjulet. I hvert spill vinner ett av tallene.

a) Lag et program i Python som simulerer ett spill med lykkehjulet.
b) Utvid programmet til å simulere 100 spill på lykkehjulet.
c) Nina kan kjøpe årer med fire tall. Hun vinner hvis lykkehjulet ender på ett av tallene på hennes åre.
   Utvid programmet til å be om fire tall som skal stå på åren. Programmet skal så simulere at lykkehjulet snurrer 100 ganger. Hver gang skal programmte skrive ut enten «tap» eller «VINST» til skjerm
d) Utvid programmet til å i stedet spille 10 000 ganger. I stedet for å skrive ut resultatet, skal programmet telle opp hvor mange ganger det ble vinst.

Videre læring

Første kursdag i kurset «Programmering i matematikk i ungdomsskolen» har fokusert på grunnleggende programmering i Python og sett på noen eksempler på anvendelser. Hvordan skal vi jobbe i tiden etter kurset for å bli bedre forberedt til å undervise i programmering?

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.
- Utforskende undervisning med programmering.
- Programmeringsdidaktikk.

https://programmeringskurs.no/

Register

Kommandooversikt

Kommando	Forklaring	Sidetall
+	addisjon	7
-	subtraksjon	7
*	multiplikasjon	7
/	divisjon	7
**	potens	7
//	heltallsdivisjon	7
%	modulo (rest)	7
<	mindre enn	14
<=	mindre enn eller lik	14
>	større enn	14
>=	større enn eller lik	14
==	lik	14
!=	ikke lik	14
and	og	14
append	legge til element (liste)	24
def	definisjon (funksjon)	22
elif	ellers hvis (if)	15
else	ellers (if)	15
False	usann	14
float	gjøre om til flyttall/desimaltall	13
for	for (løkke)	20
from	fra (import)	32
if	hvis	15

Kommando	Forklaring	Sidetall
import	importere	31
input	hente data fra brukeren	12
int	gjøre om til heltall	13
len	lengde (streng, liste)	26
not	ikke	14
math.pi	$\pi \approx 3{,}141592654$	31
or	eller	14
plt.axhline	*x*-akse	34
plt.axvline	*y*-akse	34
plt.gca	'get current axis'	35
plt.grid	rutenett	34
plt.legend	grafforklaring	35
plt.plot	tegne graf	32
plt.xlabel	navn på *x*-aksen	34
plt.xlim	*x*-intervall	34
plt.ylabel	navn på *y*-aksen	34
plt.ylim	*y*-intervall	34
print	skrive til skjerm	6
random.randint	tilfeldig heltall	37
range	tellemengde	20
return	hva funksjonen skal returnere	23
True	sann	14
while	mens (løkke)	17

Design: www.melkeveien.no og Knut Skrindo | Foto: ©Paal-André Schwital / Metro Branding