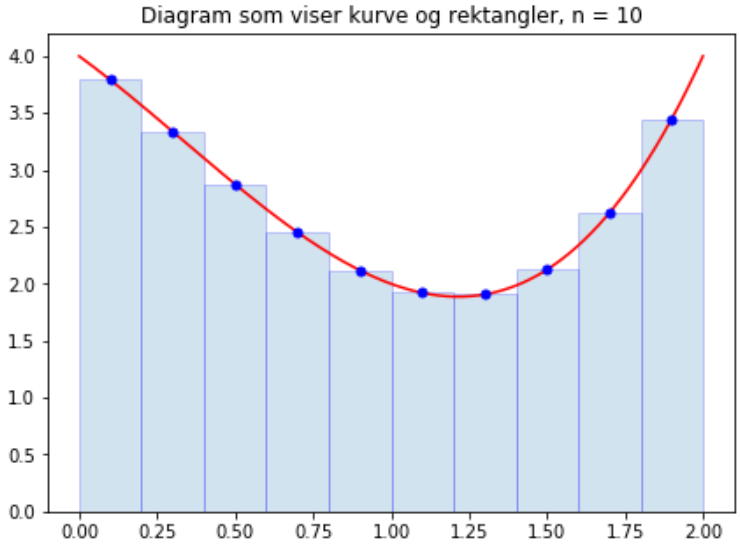
Python øvelser



av

Peer Andersen

Universitetet i Sørøst-Norge 2019

Innhold

Komme i gang med Python

Dette heftet tar for seg grunnleggende prinsipper innenfor programmering i Python. Vi vil særlig legge vekt på anvendelser mot matematikkfaget. Python er et programmeringsspråk som det er svært enkelt å komme i gang med. Selv om det er enkelt å komme i gang med Python inneholder programmet også mange funksjoner som setter oss i stand til å lage nokså avanserte programmer. Vi skal i dette heftet primært se på de grunnleggende funksjonene, men også se på noen litt mer avanserte programmer på slutten av heftet.

Det finnes mange gode kilder til Python programmering. Siden

https://www.w3schools.com/python/default.asp

inneholder mange grunnleggende eksempler på bruk av Python. Her kan du se kodene og samtidig teste ut hvordan det blir seende ut når du kjører programmet.

Websiden

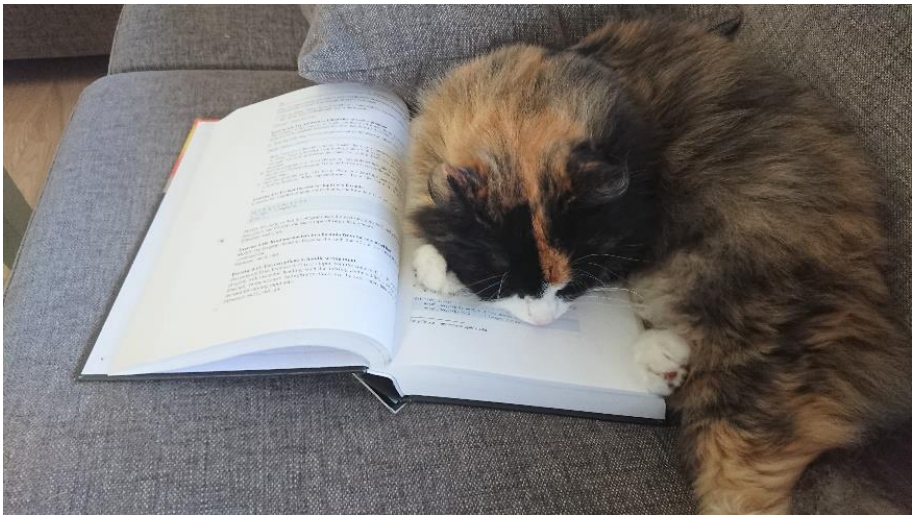
https://www.geeksforgeeks.org/python-programming-language/

tar for seg både grunnleggende prinsipper samt mer av de mer avanserte mulighetene som ligger i Python.

For å tegne figurer bruker vi mathplotlib. I lenken under finner du mye nyttig stoff om hvordan du lager diagrammer og plot i Python.

https://matplotlib.org/index.html

Boken A primer on scientific programming with python av Hans Petter Langtangen kan også anbefales. Her finner du mye nyttig og spennende stoff om hvordan Python kan anvendes i matematikk.



Alle kan lære seg Python, også pus. Her fordyper hun seg i Langtangen sin bok.

Installasjon

Jeg vil anbefale å laste ned Anaconda og deretter Spyder. Her er en liten video som viser hvordan du kan installere Anaconda og Spyder.

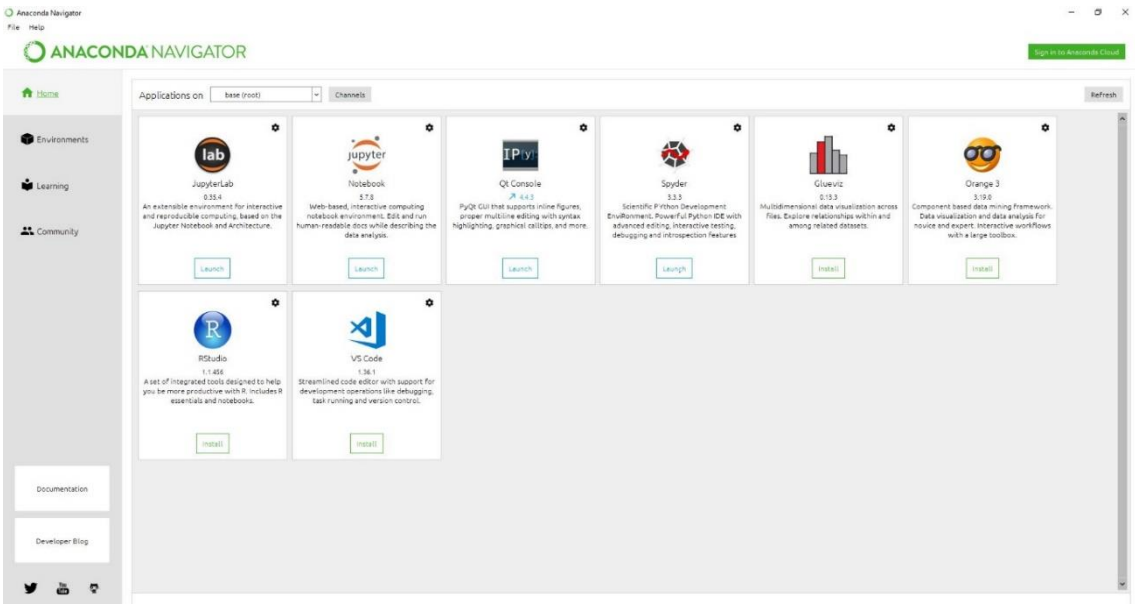
Video som viser hvordan du installerer Anaconda og Spyder

Du kan laste ned Anaconda herfra

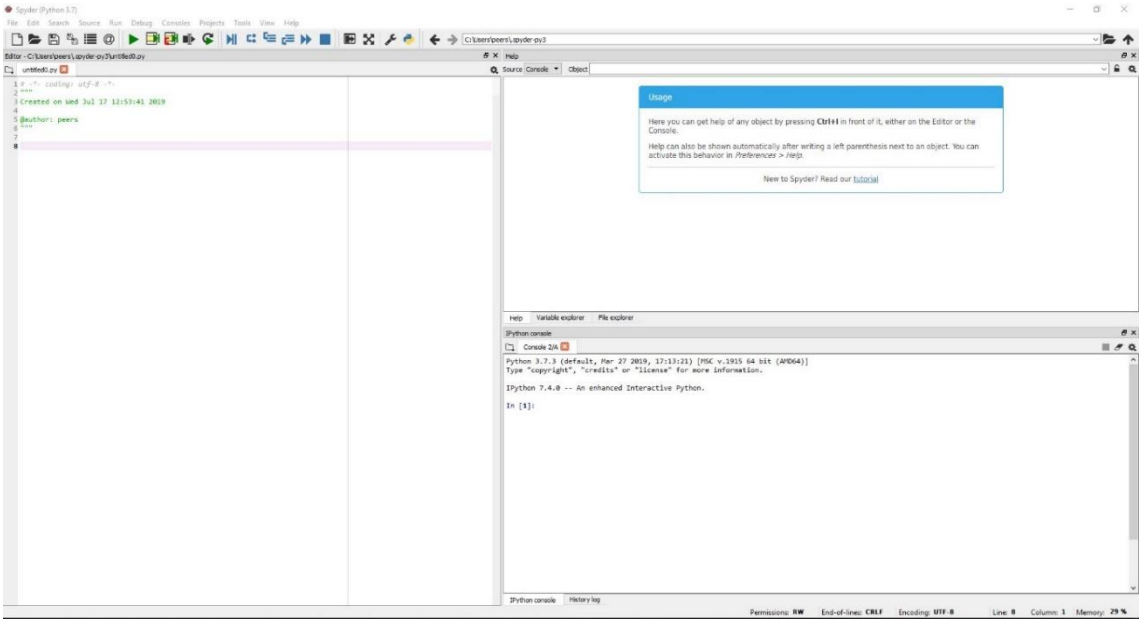
https://www.anaconda.com/distribution/

Velg Python 3.7 versjonen.

Når du har lastet ned Anaconda får du opp et vindu som vist under

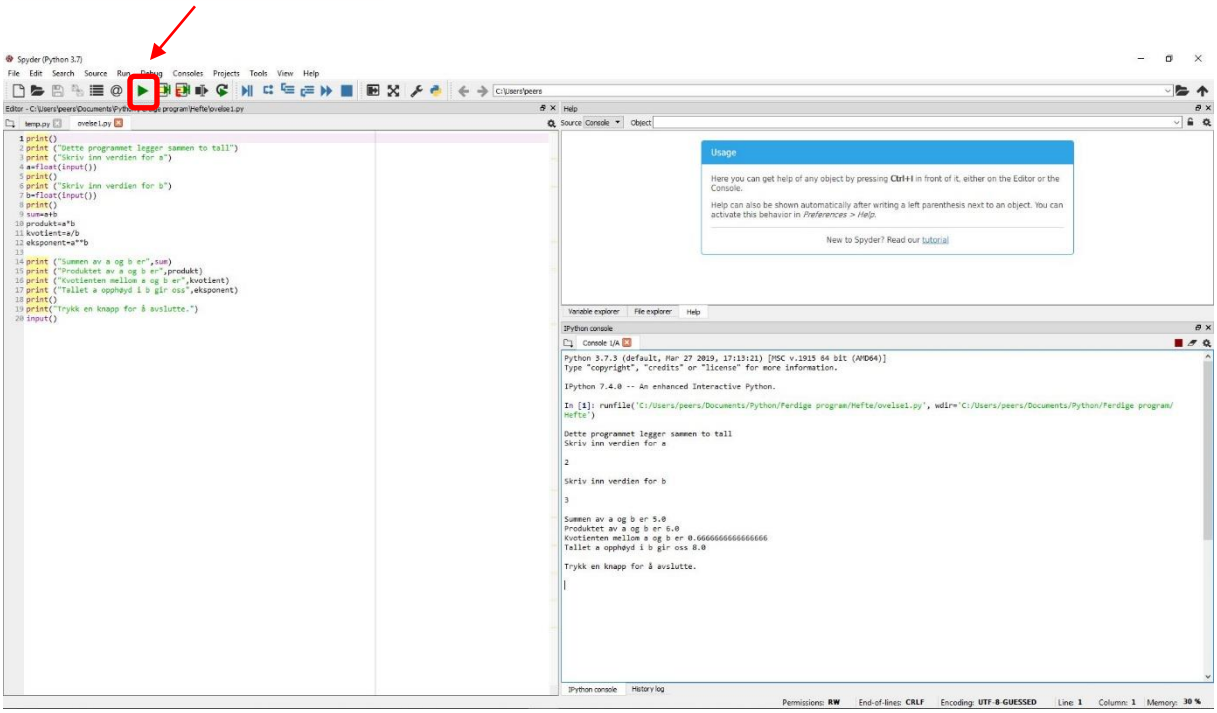


Her velger du Launch under Spyder. Da får du installert Spyder. Spyder vinduet ser ut som vist under



Du kan skrive Python koden din i vinduet til venstre. Når programmet skal kjøres klikker du på grønne pilen i menyraden og resultatet vises i vinduet som heter Console. Vinduet oppe til høyre kan du ta vekk om du ønsker mer plass til kjøringen. Klikk bare på x oppe til høyre så blir det borte. Du kan ta inn andre vindu om du ønsker det. Da går du på View og deretter på Panes og velger hva du vil vise. Når du har lagt in et program og kjørt det vil det typisk se ut som vist på neste side før du har fjernet Hjelp vinduet.

Klikk på denne knappen når du vil kjøre programmet.



Selve programkoden er skrevet i vinduet til venstre og kjøringen vises ned til høyre. Knappen med grønne pilen som brukes for å kjøre programmet er merket.

Alle programmene vi skal jobbe med finner du på en egen webside.

https://web01.usn.no/~panderse/pythonhefte.html

Heftet er bygget ved at du går gjennom et sett med øvelser. I presentasjonen er alle kommandolinjer vi skal bruke rykket inn et hakk i forhold til teksten ellers. Alle kommandoer som er nevnt i selve teksten er angitt med kursiv. Ellers er komplette program presentert i en ramme med enkel linje og kjøringene er presenter i ramme med dobbel linje.

Øvelse 1. Grunnleggende funksjoner

I denne øvelsen skal du lære mer om hvordan du kan skrive en tekst i Python, hente inn to tall og deretter legge dem sammen og skrive ut svaret.

Kommandoen *print* kan brukes når du skal legge inn tekst i programmet. Prøv å skrive inn følgende kommando

```
print ("Dette programmet leser inn to tall og legger dem sammen.")
```

Trykk deretter på Kjør knappen. Programmet skal da skrive ut teksten

```
Dette programmet leser inn to tall og legger dem sammen.
```

Det neste vi skal gjøre nå er å lese inn to tall fra skjermen. I Python er det en funksjon som heter input() som hjelper oss med dette. Vi skal først legge inn en liten tekst som forteller at vi skal skrive inn det første tallet som vi kaller for a. Du kan nå skrive inn teksten

```
print ("Skriv inn verdien for a")
```

Det neste du skal gjøre er å bruke input funksjonen. Rett etter linjen over kan du skrive inn

```
a=float(input())
```

Legg merke til at vi skriver *float* først. Det er fordi vi ønsker at maskinen skal behandle a som et reelt tall. Hadde vi skrevet *int* i istedenfor ville den betraktet det som et heltall. Dersom vi kun skriver *input()* tar maskinen det som en tekst.

Det maskinen gjør er at den gir a den verdien du skriver inn på skjermen. Det neste vi skal gjøre er å legge inn verdien for b. Vi bruke samme prinsipp som når vi legger inn a. Koden

```
print ("Skriv inn verdien for b")
b=float(input())
```

vil legge inn en verdi i b. Nå har vi fått lest inn et tall a og et tall b. Vi kan nå gå i gang å gjøre beregninger på disse tallene. Dette kan gjøres på følgende måte

```
sum=a+b
produkt=a*b
kvotient=a/b
eksponent=a**b
```

Når en skal opphøye et tall i et annet tall må en bruke ** og ikke ^ som vi er vant med fra blant annet Excel og GeoGebra. Svarene på regneoperasjonene er nå lagret i den variabelen vi har spesifisert, altså sum, produkt, kvotient og eksponent. Vi kunne selvsagt brukt andre navn, men det lønner seg å bruke noe som har et noenlunde logisk navn. Når vi har gjort disse utregningene kan vi skrive ut resultatet. Vi skal igjen bruke printfunksjonen, men på en litt annen måte. Når vi skal skrive ut summen kan vi gjøre dette på følgende måte

```
print ("Summen av a og b er",sum)
```

Det programmet gjør er det først skriver ut teksten Summen av a og b er: Deretter skriver den ut verdien som ligger i variabelen sum på samme linje. Vi skiller teksten og variabelen med et komma. Resultatet blir slik dersom vi har lagt inn 2 for a og 3 for b:

Summen av a og b er 5.0

Det er ikke noe i veien for å ha med flere variable i samme linje. Vi kunne f. eks skrevet ut sum og produkt i samme kommando

```
print ("Summen og produktet av a og b er henholdsvis", sum,"og", produkt)
```

Resultatet av denne kommandoen blir:

Summen og produktet av a og b er henholdsvis 5.0 og 6.0

Et komplett program der en leser inn a og b og deretter regner ut sum, produkt, kvotient og a opphøyd i b ser ut som vist under. Du finner hele koden i filen ovelse1.py

```
print ("Dette programmet legger sammen to tall")
print ("Skriv inn verdien for a")
a=float(input())
print()
print ("Skriv inn verdien for b")
b=float(input())
print()
sum=a+b
produkt=a*b
kvotient=a/b
eksponent=a**b

print ("Summen av a og b er",sum)
print ("Produktet av a og b er",produkt)
print ("Kvotienten mellom a og b er",kvotient)
print ("Tallet a opphøyd i b gir oss",eksponent)
```

Resultatet ser slik ut

```
Dette programmet legger sammen to tall
Skriv inn verdien for a

2

Skriv inn verdien for b

3

Summen av a og b er 5.0
Produktet av a og b er 6.0
Kvotienten mellom a og b er 0.6666666666666666
Tallet a opphøyd i b gir oss 8.0
```

Vi ser at formen på utskriften ikke er særlig pen. Blant annet får vi med mange desimaler når vi deler 2 på 3. Vi kan gjøre en del med formatteringen for å få det penere. Det kommer vi tilbake til i øvelse

9. I første omgang er det viktigste at du blir kjent med de grunnleggende prinsippene for programmering.

Python skiller mellom små og store bokstaver. Det er lett for å starte en setning med stor bokstav som f. eks å skrive Print. Gjør du det kommer det en feilmelding når du kjører programmet.

Du vil se at i kodene som presenteres utover så er det langt inn en kommando print() en del steder. Dette er en kommando for å lage et linjeskift. Det er ikke alltid den er tatt med alle steder. Når du jobber med programmene så legg selv inn linjeskift der du finner det naturlig slik at utskriften blir oversiktlig og fin.

På slutten av alle program har jeg lagt inn setningen

```
print("Trykk en knapp for å avslutte.")
```

Denne trenges strengt tatt ikke om du kun kjører programmene fra Spyder. Kjører du det på andre måter kan du risikere at programmet lukker seg når det er ferdig uten at du får sett på resultatet. Ved å bruke denne kommandoen til slutt sikrer du deg at det ikke forsvinner før du har trykket en tast.

Øvelse 2. Løkker

En av de mest nyttige funksjonene i programmering er løkker. Løkker brukes typisk i situasjoner der en skal gjenta en operasjon et visst antall ganger. Dette kan være f. eks å beregne verdien til en funksjon, simulere et visst antall terningkast etc. De to mest brukte måtene å lage løkker på i Python er ved hjelp an *for* kommandoen eller ved å bruke kommandoen *while*. Vi skal se litt på bruken av begge.

For løkker

Vi viser med et eksempel hvordan en *for* løkke fungerer. Vi skal lage en løkke som starter med x er lik 2 og går opp til x er lik 5. For hver gang vi går gjennom løkken skrives x ut. Koden under viser hvordan vi gjør dette.

```
for x in range(2, 6):
    print(x)
```

Programmet starter med å la x være 2. Deretter gjennomfører den det som står i innrykket under *for* setningen. I dette tilfelle skriver den ut 2. Deretter starter programmet forfra igjen, men nå er x økt til 3. Deretter skriver den ut 3. Slik fortsetter det helt til x=5. Programmet stopper altså for x verdien som er 1 mindre enn det som står til slutt i parentesen. Programmet over vil dermed skrive ut tallene

```
2
3
4
5
```

Vi må alltid avslutte *for* setningen med et kolon, ellers får vi feilmelding. Det som skal utføres i løkken må stå som et inntrykk. Dette er et viktig prinsipp i Python og gjelder også for *if* setninger som vi kommer til senere. I dette tilfellet har vi kun en kommando i løkken. Det er ingenting i veien for å ha flere kommandoer om vi ønsker det. Når vi har angitt startverdi og sluttverdi som i dette tilfelle så øker x verdien med 1 for hver gang programmet gjennomgår løkken. Vi kan også øke med et vilkårlig tall. Da skriver vi det til slutt i parentesen. La oss se på et programs som kombinerer litt av det vi har tatt opp. Vi skal lage et program som regner ut antall grader Fahrenheit når vi kjenner antall grader Celsius. Vi skal angi temperaturen vi starter med, temperaturen vi skal slutte med og hvor mye vi skal øke med for hver gang. Deretter skal programmet regne temperaturene og skrive det pent ut. Under ser du et komplett program som håndterer dette. (ovelse2a.py)

```
print()
x=int(input("Skriv inn temperaturen vi skal starte med : "))
y=int(input("Skriv inn temperaturen vi skal stoppe med : "))
n=int(input("Skriv deretter inn spranget mellom temperaturene : " ))
print()
for i in range(x,y,n):
    f=1.8*i+32
    print(i,"grader Celsius tilsvarer",f,"grader Fahrenheit")
```

Vi skal se på hva dette programmet gjør. I den første linjen legger vi inn starttemperaturen x. Vi har her gjort en vri fra første øvelsen ved at vi setter inn teksten i parentesen etter input. Det gjør at vi sparer en linje samtidig som vi da kan skrive inn verdien rett etter teksten. I andre linjen legger vi inn

sluttemperaturen y og til slutt hvor mye vi øker med for hver gang som n. Deretter kommer *for* løkken. Vi ser her at vi istedenfor tall bruker variablene vi nettopp har lest inn verdiene til. Det neste vi gjør er å regne ut antall grader Fahrenheit når vi kjenner antall grader Celsius. Den matematiske formelen

$$f(x) = 1{,}8x + 32$$

gjør denne omregningen. I vårt tilfelle så vil variabelen i tilsvare temperaturen i Celsius grader. Vi lager derfor en formel som beregner hva dette tilsvarer i Fahrenheit. Resultatet tilordner vi variabelen f. Det neste vi skal gjøre er å skrive ut resultatet. Vi bruker som før *print* setningen til dette. Vi ser at i vårt tilfelle har vi med fire ledd. Først skriver vi temperaturen i Celsius, deretter en tekst som står i anførselstegn. Deretter får vi temperaturen i Fahrenheit representert med variabelen f og til slutt tekst igjen. Vi ser at for hver gang programmet går gjennom løkken så gjør den to operasjoner. Først beregnes f og deretter skriver den ut resultatet. Om du kjører programmet og legger inn 0 som startverdi, 21 som sluttverdi og steg på 4 så skal resultatet bli som vist under. (Vi legger inn 21 som sluttverdi slik at vi får med oss 20. Husk programmet stopper på 1 mindre enn det som legges inn som sluttverdi.)

```
Skriv inn temperaturen vi skal starte med: 0

Skriv inn temperaturen vi skal stoppe med: 21

Skriv deretter inn spranget mellom temperaturene: 4

0 grader Celsius tilsvarer 32.0 grader Fahrenheit
4 grader Celsius tilsvarer 39.2 grader Fahrenheit
8 grader Celsius tilsvarer 46.4 grader Fahrenheit
12 grader Celsius tilsvarer 53.6 grader Fahrenheit
16 grader Celsius tilsvarer 60.8 grader Fahrenheit
20 grader Celsius tilsvarer 68.0 grader Fahrenheit
```

Det finnes mange andre muligheter med *for* løkker enn det vi har presentert her. Har du lært det vi har gått gjennom her har du imidlertid nok kunnskap til å lage enkle programmer med *for* løkker.

while løkker

En annen teknikk vi kan bruke for å lage løkker er *while* kommandoen. Den er litt annerledes enn *for* løkken. Vi skal her se på et enkelt eksempel på hvordan den fungerer ved å studere programmet under.

```
n=int(input("Vi skal kvadrere de naturlige tallene opp til n. Legg inn n : "))
print()
i=1
while i<n+0.1:
   x2=i**2
   print("Når x=",i,"så er x^2=",x2)
   i=i+1
print()
```

Dette programmet regner ut kvadratet av de n første naturlige tallene. I linje 1 legger vi inn antall tall vi skal kvadrere. Vi legger deretter inn 1 som verdi for i siden vi skal starte med 1 som er det første naturlige tallet. Det *while* setningen nå gjør er at den sammenlikner i med n+0.1 og dersom i<n+0.1 så utfører den det som står i innrykket. Vi ser først at den regner ut i^2 og legger resultatet inn i variabelen x2. Deretter skriver den ut x verdien og x^2 med litt tekst. Til slutt kommer en setning som matematisk sett er helt feil $i = i + 1$, men som brukes mye innfor programmering. Det programmet gjør er at den tar verdien for i og legger til 1 og deretter kaller resultatet for i. Resultatet er at variabelen i, i dette tilfellet øker med 1. Programmet hopper deretter opp til *while* setningen og sjekker om i (som nå har økt med 1) er mindre enn n+0.1. Dette gjentas helt til vilkåret ikke lenger er oppfylt. Da hopper programmet videre til neste steg i programmet. I

Som du kanskje har lagt merke til så sjekker den i mot n+0.1 og ikke bare n. Det er fordi vi ønsker at n skal være med og da legger vi til et lite tall for å være sikker på at n kommer med. Under ser du resultatet av kjøringen når vi har valgt n lik 5

```
Vi skal kvadrere de naturlige tallene opp til n. Legg inn n: 5

Når x= 1 så er x^2= 1
Når x= 2 så er x^2= 4
Når x= 3 så er x^2= 9
Når x= 4 så er x^2= 16
Når x= 5 så er x^2= 25
```

I mange tilfeller kan en bruke både *for* setning og *while* setning for å løse et problem. Hva en foretrekker er litt smak og behag. I videre fremstillingen vil vi bruke begge deler. Vi kommer mer tilbake til både *for* løkker og *while* løkker i de videre øvelsene. Koden finner du i filen ovelse2b.py.

Øvelse 3. If setninger

En annen funksjon som er svært nyttig er *if* setningen. Den gir oss muligheten til å undersøke om et vilkår er oppfylt eller ikke. Dersom det er oppfylt så utføres kommandoene vi har angitt. La oss belyse det med et eksempel. (ovelse3a.py)

```
a=int(input("Skriv inn et tall a:"))
b=int(input("Skriv inn et tall b:"))
if a>b:
     print("a er større en b")
else:
    print("b er større eller lik a")
```

Programmet starter med at vi leser inn et heltall a og et heltall b. Vi skal deretter avgjøre om hvilket tall som er størst. Vi bruker *if* setningen til det. Setningen sjekker for oss om a er større enn b. Dersom dette er tilfelle utfører den kommandoen med innrykk, altså den skriver at a er større enn b. Husk at du også må avslutte *if* setningen med et kolon. Dersom dette ikke er tilfelle, så utfører den det som står som innrykk etter *else* kommandoen. I dette tilfelle er det at b er større eller lik a. Dersom vi legger inn at a=2 og b=3 vil vi få frem følgende bilde.

```
Skriv inn et tall a: 2

Skriv inn et tall b: 3

b er større eller lik a
```

I Python er de logiske operatorene som vist under. Legg særlig merke til kommandoene for å sjekke om to tall er like samt kommandoen for at to tall er forskjellige. Disse skiller seg litt fra blant annet Excel.

Lik: a==b
Ikke like: a !=b
Mindre enn: a<b
Mindre enn eller lik: a<=b
Større enn: a>b
Større enn eller lik: a>=b

Som du ser så klarer ikke vårt program å sjekke om a er lik b. Vi kan enkelt utvide programmet til å gjøre det også. Da bruker vi en kommando som heter *elif* i tillegg til det vi har gjort. Vi viser også dette gjennom et eksempel. (ovelse3b.py)

```
a=int(input("Skriv inn et tall a: "))
b=int(input("Skriv inn et tall b: "))
print()
if a>b:
   print("a er større en b")
elif a==b:
   print("a er lik b")
else:
    print("b er større enn a")
```

Det programmet nå gjør er at det først sjekker om a er større enn b. Om det er tilfelle skriver det ut at a er større enn b. Dersom dette ikke er oppfylt går det videre og gjør en ny sjekk med *elif* kommandoen. I vårt tilfelle sjekker programmet om a er lik b. Dersom det er tilfelle skriver det ut at a er lik b. Om heller ikke dette vilkåret er oppfylt, så skriver det ut at b er større enn a.

I vårt tilfelle har vi sjekket om et vilkår er oppfylt. Vi kan også sjekke om to vilkår begge er oppfylt med kommandoen *and* eller om minst ett vilkår er oppfylt med *or* kommandoen. Vi ser også her på et eksempel. (ovelse3c.py)

```
b1=int(input("Skriv alder på barn 1: "))
k1=input("Skriv inn kjønnet på barn 1 (g/j): ")
b2=int(input("Skriv alder på barn 2: "))
k2=input("Skriv inn kjønnet på barn 2 (g/j): ")
print()

if k1=="g" and k2=="g":
  print("Begge barna er gutter")

if k1=="j" or k2=="j":
  print("Minst et av barna er jente")

if k1=="j" and b1>=10:
  print("Barn 1 er en jente på 10 år eller mer")
elif k1=="g" and b1>=10:
   print("Barn 1 er en gutt på 10 år eller mer")
else:
   print("Barn 1 er under 10 år.")
```

I dette programmet skal vi først lese inn alder og kjønn til to barn. Vi kaller disse variablene for b1 og b2 (alder) og k1 og k2 (kjønn). Alder legger vi inn som et heltall ved å bruke *int*, mens kjønn skal være en tekst og derfor står det ikke noe før *input*. Når vi har lagt inn disse verdiene skal vi gjøre noen tester. Det første vi skal sjekke er om begge er gutter. Vi bruker som før *if* setningen til det. Vi sjekker om k1 er lik g ved å skrive inn k1=="g". Legg merke til at vi bruker "" tegn når vi skal sjekke om noe er lik en tekst. Vi bruker deretter kommandoen *and* og deretter sjekker vi om k2 også er lik g. Når vi bruker *and* kommandoen så må begge vilkårene være oppfylt for at programmet skal utføre kommandoen i innrykket. Programmet fortsetter deretter med å sjekke om k1=="j" og om k2=="j". Vi ser her at vi har *or* mellom testene, som betyr at vi utfører det som står i innrykket om minst ett av vilkårene er oppfylt. I istedenfor denne *if* setningen kunne vi også bruke *else* på testen om begge er gutter. Det ville gjort samme nytte. Ofte i Python så kan ting gjøres på mange ulike måter.

I siste delen av programmet sjekker vi først om barn 1 er jente og om barnet er 10 år eller eldre. Om begge disse vilkårene er oppfylt så skriver den at

> Barn 1 er en jente på 10 år eller mer

Om det ikke er tilfelle gjennomføres en ny test. Vi sjekker deretter om barn 1 er gutt på 10 år eller mer. Dersom dette er oppfylt skriver programmet ut setningen

> Barn 1 er en gutt på 10 år eller mer

Om heller ikke det vilkåret er oppfylt konkluderer programmet med at Barn 1 er under 10 år.

Dersom vi legger inn at barn 1 er jente på 12 år og barn 2 er gutt på 8 år får vi følgende resultat

```
Skriv alder på barn 1: 12

Skriv inn kjønnet på barn 1 (g/j): j

Skriv alder på barn 2: 8

Skriv inn kjønnet på barn 2 (g/j): g

Minst et av barna er jente
Barn 1 er en jente på 10 år eller mer
```

Vi kunne lagt inn flere spørringer enn det vi har gjort. Dette kan du teste ut selv.

Det er selvsagt ikke noe i veien for å legge inn flere *and* eller *or* kommandoer. Det er også mulig å kombinere disse.

Øvelse 4. Løsning av andregradslikningen

I denne øvelsen skal vi se hvordan vi kan bruke Python til å løse en andregradslikning. Vi skal bruke det vi lærte i forrige øvelse om *If* setninger. Vi skal også introdusere noen nye elementer i denne øvelsen.

Likningen vi skal løse er andregradslikningen

$$ax^2 + bx + c = 0$$

Vi kan vise at løsningen kan skrive som

$$x = \frac{-b \mp \sqrt{b^2 - 4ac}}{2a}$$

I programmet vårt skal vi lese inn verdiene a, b og c og deretter skal programmet regne ut løsningene for oss. Om det ikke er løsning skal den informere oss om det. Vi starter med å legge inn litt tekst og kommandoen for å lese inn flere verdier

```
from math import sqrt
print()
print("Dette programmet løser andregradslikningen ax^2+bx+c=0")
a, b, c =[float(x) for x in input("Skriv inn a, b og c (skill dem med mellomrom): ").split()]
```

I Python er det behov for å importere matematiske funksjoner fra math biblioteket. I vårt tilfelle skal vi bruke kvadratrot funksjonen og vi starter derfor med å importere den. Vi skal se nærmere på kommandoen der vi legger inn verdier for a, b og c. I utgangspunktet er kommandoen for dette

```
a, b, c=input("Skriv inn a, b og c (skill dem med mellomrom): ").split()
```

Når vi tar med split() til slutt vil verdiene vi legger inn splittes opp og legges inn i variablene vi har skrevet inn. Når dette gjøres legges verdiene inn som en tekststreng uten at det gir oss mulighet til å utføre beregninger på dem. For å unngå dette tar vi med klammeparentesene og presiserer at verdiene skal være desimaltall gjennom å bruke float slik det er vist.

Det neste vi skal gjøre er å regne ut løsningene. Vi starter med å regne ut uttrykket som står under kvadratrottegnet. Vi kaller det for rot.

```
rot=b**2-4*a*c
```

Dette gjør vi for å spare litt skrivearbeid og gjøre programmet litt mer oversiktlig. Når vi har gjort dette er vi klare til å skrive inn resten av koden. Det første vi skal gjøre er å sjekke om a=0. Om det er tilfelle vil likningen ikke lenger være en andregradslikning men førstegradslikningen

$$bx + c = 0$$

Denne likningen har løsningen

$$x = -\frac{c}{b}$$

Om det er tilfelle skrives løsningen ut. Om dette ikke er en førstegradslikning fortsetter programmet med å beregne løsningen til andregradsuttrykket. Koden vi skal bruke er vist på neste side.

```
if abs(a)<0.000000001:
   print("Dette er en førstegradslikning med løsningen x =",-c/b)
elif rot>=0:
   x1=(-b-sqrt(rot))/2/a
   x2=(-b+sqrt(rot))/2/a
   if abs(x2-x1)<0.0000001:
      print("De er kun en løsning x = ", x1)
   else:
      print("Løsning en er x1 =", x1)
      print("Løsning to er x2 =",  x2)
else:
   print("Likningen har ingen løsning")
```

Vi starter med å sjekke om a er lik 0. Siden dette er desimaltall kan avrundinger gjøre at selv om a er lik 0 så vil den få en bitteliten verdi som f. eks 0,00000000000000001. Det er derfor uheldig å sjekke om a er eksakt lik 0. Vi kunne gjort det om a var definert som et heltall. Vi velger her istedenfor å sjekke om absoluttverdien til a er mindre enn en liten verdi. Er det tilfelle konkluderer vi med at vi har en førstegradslikning og da skriver vi ut hva løsningen blir.

Hvis dette ikke er tilfelle går vi videre med en *elif* setning. Det vi sjekker her er om uttrykket under rottegnet er større eller lik 0. Om det er tilfelle har vi løsning. Løsningene vil være x1 og x2 som er står i innrykket etter *elif* setningen. Vi kan komme i en situasjon der vi kun har en løsning. Det skjer om uttrykket under rottegnet er lik 0. Likningen $x^2 - 4x + 4 = 0$ har f. eks kun en løsning som er $x = 2$. Vi sjekker om det er tilfelle ved å bruke en *if* setning. Vi undersøker om løsningene er like eller ikke. Også her ser vi på om absoluttverdien til differansen til løsningene er mindre enn en lite tall for å ta høyde for avrundingsfeil. Om dette er tilfelle skriver vi ut at det kun er en løsning og angir denne. Siden x1 og x2 er lik i det tilfelle skriver vi ut en av dem. Det er i grunnen det samme hvilken av dem du skriver ut. Om løsningene ikke er like bruker vi *else* setningen til å skrive ut begge løsningene. Helt til slutt benytter vi enda en *else* setning. Dette er for å angi hva vi skal gjøre om uttrykket under rottegnet er negativt. Da skal vi skrive inn at vi ikke har løsning.

Kjører du hele programmet (som du også finner i ovelse4a.py) og velger a=1, b=5 og c=4 får du

```
Dette programmet løser andregradslikningen ax^2+bx+c=0

Skriv inn a, b og c (skill dem med mellomrom): 1 5 4

Løsning en er x1 = -4.0
Løsning to er x2 = -1.0
```

<u>Komplekse løsninger</u>

Python gir oss også muligheter til å regne med komplekse tall. Om du ikke er fortrolig med komplekse tall så hopp bare over dette avsnittet.

Vi skal se nærmere på hvordan vi kan angi løsningene om vi får komplekse løsninger av likningen. Setningen

```
print("Likningen har ingen løsning")
```

erstatter vi med

```
from cmath import sqrt
x1=(-b-sqrt(rot))/2/a
x2=(-b+sqrt(rot))/2/a
print("Likningen har to komplekse løsninger")
print()
x1 = complex(round(x1.real,3),round(x1.imag,3))
x2 = complex(round(x2.real,3),round(x2.imag,3))
print("Løsning en er z1 =",x1)
print("Løsning en er z1 =",x2)
```

Vi starter med å importere funksjonen sqrt fra cmath som er en modul med komplekse funksjoner. Når vi gjør det, vil vi kunne regne med komplekse tall. Løsningen x1 og x2 som nå beregnes vil være komplekse løsninger. I Python brukes j og ikke i for å vise den imaginære delen. Et komplekst tall som vi ville skrevet som f. eks 3+2i vil Python skrive som 3+2j. Deretter kommer en setning der vi skriver at vi har komplekse løsninger. Det neste vi skal gjøre er å runde antall desimaler av til 3.

Kommandoen *round* runder av et tall til et gitt antall desimaler.

```
round(sqrt(2),3)
```

Vil f. eks runde av kvadratroten til 2 til 3 desimaler.

Å runde av komplekse tall er litt tungvint siden vi ikke kan avrunde x1 i en operasjon. Det vi gjør her er å runde av reelle delen og komplekse delen hver for seg. Dette setter vi sammen til et nytt komplekst tall. Til slutt skriver vi ut den komplekse løsningen.

Legger vi inn a=1, b=2 og c=5 vil vi få følgende resultat

```
Dette programmet løser andregradslikningen ax^2+bx+c=0

Skriv inn a, b og c (skill dem med mellomrom): 1 2 5

Likningen har to komplekse løsninger

Løsning en er z1 = (-1-2j)
Løsning en er z1 = (-1+2j)
```

Hele programmet finner du som ovelse4b.py

Øvelse 5. Lister

Lister er noe som er svært nyttig i Python. Vi illustrerer hvordan lister fungerer gjennom noen eksempler. En liste kan f. eks inneholde navn eller tall som f. eks dette

```
navn=["Ole","Kari", "Lise"]
tall=[2,4,6,8,10]
```

En liste skrives på formen som vist over der to klammeparenteser omslutter listen med elementer. Elementene skiller vi med et komma. Når vi har laget listen kan vi gjøre en rekke operasjoner med de. Vi kan f. eks skrive ut et bestemt element i listen. Kommandoen

```
print(navn[1])
```

skriver ut Kari som resultatet. Legg merke til at første posisjonen i listen er 0, slik at Kari står på posisjon 1. Dette er viktig å huske og lett å glemme, i alle fall i starten. Skriver vi kommandoen

```
print(tall[0])
```

vil den gi 2 som svar. Vi kan skrive ut hele listen. Da skriver vi bare navnet på listen uten noe klammeparentes. Kommandoen

```
print(navn)
```

gir oss resultatet

```
['Ole', 'Kari', 'Lise']
```

Vi kan enkelt endre verdi av et element i listen. La oss si at vi skal bytte ut Lise med Siv. Det fikser vi med kommandoen

```
navn[2]="Siv"
```

Prøver du å skrive ut *print(navn)* vil du få ut resultatet

```
['Ole', 'Kari', 'Siv']
```

Vi skal se nærmere på noen andre nyttige egenskaper med lister.

<u>Legge til og fjerne elementer fra listen</u>

Ofte er det behov for å kunne legge til elementer i listen. Det kan gjøres enkelt med å bruke en kommando som heter *append*. Kommandoen

```
navn.append("Petter")
```

vil legge til Petter til slutt i listen. Resultatet om du printer ut listen blir da

```
['Ole', 'Kari', 'Siv', 'Petter']
```

Noen ganger er det behov for å legge til et element på en bestemt plass i listen. La oss si at vi ønsker å legge til Arild mellom Kari og Siv. Det får vi til ved å bruke en kommando som heter *insert*. Den brukes som vist under

```
navn.insert(2, "Arild")
```

Resultatet blir at Arild legges inn på posisjon 2 i listen, som altså er element 3. Resultatet om vi printer ut listen blir

```
['Ole', 'Kari', 'Arild', 'Siv', 'Petter']
```

Det er også mulig å fjerne elementer i lisen. Det gjøres med kommandoen *pop()*. Om vi ikke spesifiserer noe i parentesene fjernes siste elementet.

```
tall.pop()
```

tar bort 10 tallet fra listen som heter tall. Resultatet om du skriver

```
navn.pop()
print(tall)
```

bør bli dette

```
[2, 4, 6, 8]
```

Ønsker vi f. eks å ta vekk 4 tallet fra listen så er det fullt mulig ved å spesifisere dette i parentesen på pop funksjonen.

```
tall.pop(1)
print(tall)
```

gir resultatet

```
[2, 6, 8]
```

Lengde på lister

Ofte kan det være nyttig å beregne lengde på en liste. Kommandoen

```
lengde=(len(navn))
```

vil beregne antall elementer i listen som heter navn. Etter at vi har lagt til Petter og Arild vet vi at listen teller 5 navn og variabelen lengde vil dermed ha verdi 5. Vi kan teste dette ved å utføre kommandoene og deretter be maskinen skrive ut resultatet.

```
print("Antall navn i listen som heter navn er: ",lengde)
```

Resultatet av dette bør bli

```
Antall navn i listen som heter navn er: 5
```

Spørringer i en liste

Det er mulig å sjekke om et bestemt element er med i en liste eller ikke og samtidig be maskinen utføre en bestemt kommandoen om det faktisk er tilfelle. Vi skal sjekke om Arild er med i listen og skrive ut en setning om at han er med i listen om det er tilfelle og i motsatt fall skrive ut setningen om at han ikke er med i listen.

```
if "Arild" in navn:
    print("Ja, Arild er i navnelisten")
else:
    print("Nei, Arild er ikke med i navnelisten")
```

Siden Arild er med i listen vil vi denne gang ut resultatet

Ja, Arild er i navnelisten

Programmet med alle kommandoene som er beskrevet her finner du i ovelse5.py. Hele programmet ser nå slik ut

```
navn=["Ole","Kari","Lise"]
tall=[2,4,6,8,10]

print(navn[1])
print(tall[0])
print(navn)

navn[2]="Siv"
print(navn)

navn.append("Petter")
print(navn)

navn.insert(2, "Arild")
print(navn)

tall.pop()
print(tall)

tall.pop(1)
print(tall)

lengde=(len(navn))
print("Antall navn i listen som heter navn er: ",lengde)

if "Arild" in navn:
   print("Ja, Arild er i navnelisten")
else:
   print("Nei, Arild er ikke med i navnelisten")
```

Resultatet bør se omtrent slik ut

```
Kari
2
['Ole', 'Kari', 'Lise']
['Ole', 'Kari', 'Siv']
['Ole', 'Kari', 'Siv', 'Petter']
['Ole', 'Kari', 'Arild', 'Siv', 'Petter']
[2, 4, 6, 8]
[2, 6, 8]
Antall navn i listen som heter navn er:  5
Ja, Arild er i navnelisten
```

Gå gjerne gjennom programkoden "for hånd" og se at du henger med på de ulike stegene.

Øvelse 6. Funksjoner og tegning av grafer

I Python er det mulig å definere funksjoner og deretter bruke dem videre i programmet. Det er også mulig å lese inn funksjonsuttrykk ved å bruke input setningen. Vi skal først se litt på hvordan funksjoner fungerer i Python. Deretter skal vi se på hvordan vi lese dem inn ved hjelp av input funksjonen og til slutt skal vi tegne opp grafen. Det første vi må gjøre om vi skal jobbe med funksjoner i Python er å importer dem fra math modulen. Det gjør vi med kommandoen

```
from math import *
```

Da får vi importert det vi trenger av funksjoner. Skal vi f. eks beregne cosinus til 0 kan vi skrive det direkte i programmet. Kommandoen

```
print("cosinus(0)=",cos(0))
```

gir 1 som svar siden cos(0)=1. De trigonometriske funksjonene regner i radianer, så husk å oppgi vinkler i radianer om du skal gjøre beregninger på dem. Ofte er det behov for å sette sammen ulike funksjoner og ikke bare bruke standardfunksjonene. Et enkelt eksempel på det kan være funksjonen $f(x) = x^2 - 4x + 3$. Vi kan definere dette som en egen funksjon i Python. Det gjøres på følgende måte

```
def f(x):
    return x**2-4*x+3
print("Når x=2 vil x^2-4x+3 være lik",f(2))
```

Den første linjen forteller hva vi skal kalle funksjonen. Jeg har kalt den for f i dette eksempelet og jeg har brukt x som variabel. Vi kan gjerne bruke andre navn eller betegnelse på variabelen. Det er heller ikke noe i veien for å bruke flere variabler. I linjen som starter med *return* bestemmer vi uttrykket for f(x). Vi må sette return først for å fortelle at f(x) skal være lik $x^2 - 4x + 3$. Når vi bruker *print* kommandoen som er angitt skriver programmet først ut en tekst og deretter beregnes $f(2)$ som i dette tilfellet blir -1.

I dette eksempelet har vi lagt inn selve funksjonen i programmet, men det er ingenting i veien for at den leses inn under kjøringen av programmet. La oss se hvordan det gjøres.

```
formel=input("Skriv inn funksjonen du skal bruke: ")
def g(x):
    return eval(formel)
print("Resultatet av g(2) blir ",g(2))
```

Det som skjer her er at funksjonen leses inn som en tekststreng og tilordnes variabelen formel. Vi definerer deretter en funksjon g(x). Vi ser her at en kommando som heter *eval(formel)* er brukt. Det *eval* kommandoen gjør er at den lar oss utføre en tekst som en Pythonkommando. I vårt tilfelle betyr det i praksis at funksjonen g(x) settes lik den innleste funksjonen som ved hjelp av *eval* funksjonen er gjort om til et matematisk uttrykk. Vi tester den deretter på g(2) og skriver ut resultatet. Programmet som er skissert over vil gi ut følgende tekst om du legger inn funksjonen $x + x^2$.

```
Skriv inn funksjonen du skal bruke: x+x**2
Resultatet av g(2) blir  6
```

Vi ser at vi først leser inn funksjonen og deretter får vi beregnet resultatet. Vi ser at det stemmer siden $2 + 2^2 = 6$.

Når du først har laget funksjonen kan du kalle den opp så mange ganger du ønsker i programmet. Det skal vi se er nyttig når vi skal tegne grafer. Det som er gjort så langt finner du i ovelse6a.py

Tegning av graf

Vi er nå klare til å tegne opp grafen. Vi starter med å importere modulene vi trenger. Det er pakken math. I tillegg skal vi importere matplotlib som gjør oss i stand til å tegne grafer og diagram. Vi skal også importere pakken numpy som er nyttig i forbindelse lister og tabeller. Disse gir vi alias np og plt og vi må bruke det som prefiks senere. Vi skal også legger inn kommandoen som gjør oss i stand til å lese inn funksjonen vi ønsker å tegne.

```
from numpy import *
import matplotlib.pyplot as plt
from math import *

formel=input("Skriv inn funksjonen du skal bruke: ")
def g(x):
    return eval(formel)
```

Vi har dermed definert funksjonen vi skal tegne som g(x). Det neste vi skal gjøre er å legge inn startverdien for funksjonen og sluttverdien. Vi legger dette i variablene a og b ved hjelp av *input* setninger

```
a=float(input("Hva er startverdien? "))
b=float(input("Hva er sluttverdien? "))
```

Det neste vi skal gjøre er å bestemme hvilke x verdier vi skal beregne funksjonsuttrykket for. Til det bruker vi kommandoen *np.linspace(a,b,n).* Den fungerer slik at vi legger inn start og sluttpunkt a og b og hvor mange punkter vi skal regne ut som n. Du kan teste denne ved å skrive inn funksjonen

```
x = np.linspace(0,10,11)
print(x)
```

Det gir resultatet

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

Legg merke til at resultatet blir skrevet som en liste. Når vi skal tegne funksjonen bør vi ha mer enn 11 punkter. Dagens datamaskiner regner raskt, så det er ikke noe problem å ta med 101 punkter eller 1001 punkter. Vi velger 101 her. Da kan vi etter å ha lest inn a og b skrive inn kommandoen

```
x = np.linspace(a,b,101)
```

Det neste vi skal gjøre er å legge inn tilhørende y-verdier fra funksjonen i en liste som vi kaller for y. Vi starter med å definere en tom liste y ved hjelp av kommandoen

```
y=[]
```

Deretter lager vi en løkke som vist under.

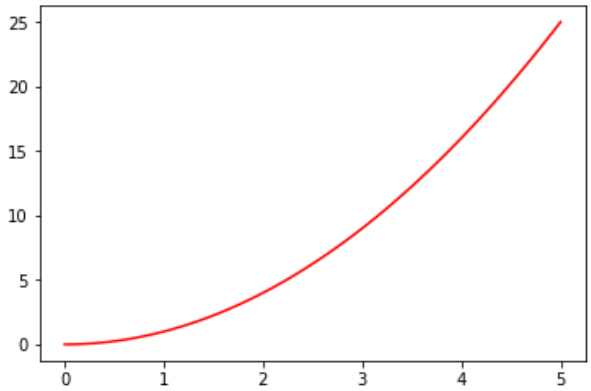
```
for i in x:
    y.append(g(i))
```

Det som skjer her at programmet går gjennom en løkke. Mer presist så starter den med å velge i=a og deretter velges i ved å ta neste element i listen helt til hele listen er gjennomgått. For hver gang den går gjennom listen, så beregnes g(i) og i tillegg legges dette som nytt element i listen y. Kommandoen *plt.plot(x,y,'r-')* tegner så opp grafen.

```
plt.plot(x,y,'r-')
plt.show
```

Legg merke til at den tar lister som argument. Teksten 'r-' indikerer at vi skal tegne den med rødt og at grafen skal tegnes som en strek.

Kommandoen *plt.show()* trengs for å få frem programmet om du kjører det rett fra terminalen, men den er ikke nødvendig om du kjører det fra Spyder.



Hele programmet finner du i filen ovelse6b.py.

Vi har tegnet opp grafen som en linje med rød farge. Det er imidlertid mange andre muligheter. Du kan f. eks velge farger ved å erstatte r med forbokstaven for fargen du vil bruke (på engelsk). Andre bokstaven eller tegnet indikerer hvilken graf du vil ha. Hos oss har vi brukt – som indikerer en linje mellom punktene. Du kunne også f. eks brukt

o for sirkel

s for et lite kvadrat

^ for en liten trekant

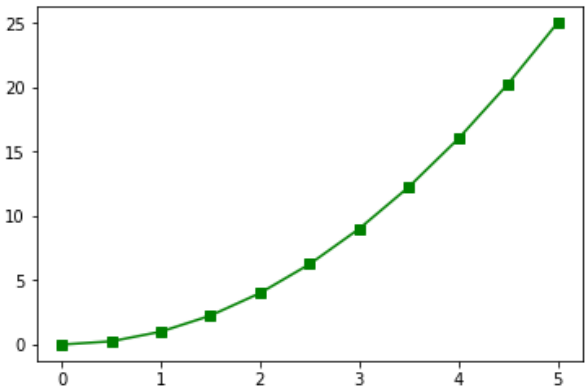
Hadde du i stedet skrevet

```
plt.plot(x,y,'g^')
```

ville programmet tegnet grønne trekanter i istedenfor en linje. Prøv gjerne dette. Når du tester ut kan du gjerne endre antall punkter fra 101 til 11 slik at det kommer tydeligere frem. Det er også mulig å tegne inn en linje og trekanter eller kvadrat i samme koordinatsystem. Det kan gjøres ved å bruke to plt.plot kommandoer.

```
plt.plot(x,y,'gs')
plt.plot(x,y,'g-')
```

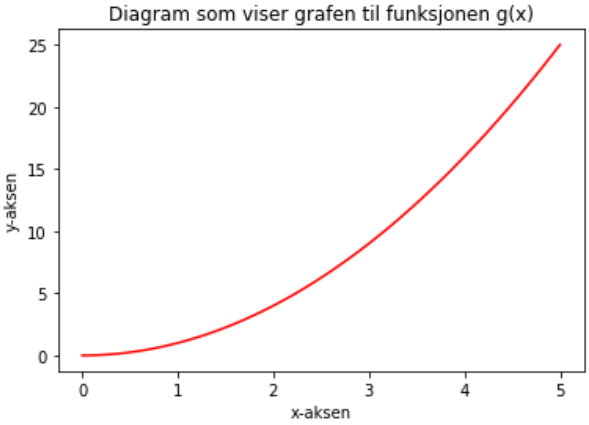
Da vil programmet først tegne opp kvadratene og deretter grafen som en linje. Bruker vi funksjonen $g(x) = x^2$ vil grafen se ut som under om vi velger 0 som startverdi og 5 som sluttverdi.



Vi kan naturligvis også legge tekst for x og y aksen samt en diagramtittel. Kommandoene hjelper oss med det.

```
plt.xlabel('x-aksen')
plt.ylabel('y-aksen')
title('Diagram som viser grafen til funksjonen g(x)')
```

Disse kommandoene legger du inn før du plt.plot kommandoen. Går vi tilbake til en rød linje og 101 punkter så vil grafen for $g(x) = x^2$ fra 0 til 5 se slik ut.



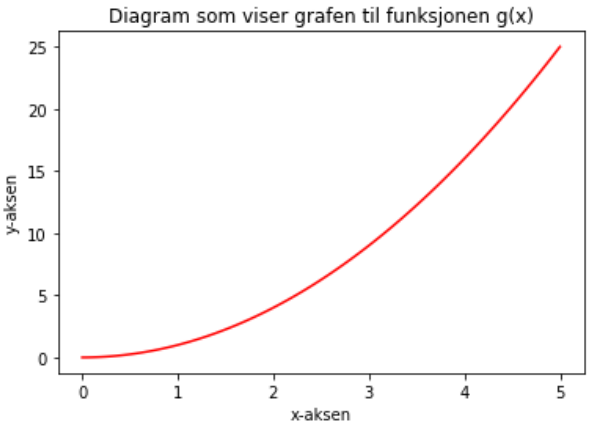
Hele programmet ser ut som vist under

```
from numpy import *
import matplotlib.pyplot as plt
from math import *

formel=input("Skriv inn funksjonen du skal bruke: ")

def g(x):
    return eval(formel)

a=float(input("Hva er startverdien? "))
b=float(input("Hva er sluttverdien? "))

x = np.linspace(a,b,101)
y=[]
```

```
for i in x:
    y.append(g(i))

plt.xlabel('x-aksen')
plt.ylabel('y-aksen')
title('Diagram som viser grafen til funksjonen g(x)')
plt.plot(x,y,'r-')
plt.show()
```

Som du ser så er det ikke en lang og avansert kode for å legge inn funksjonen og deretter tegne den. Programmet vil gi følgende resultat

```
Skriv inn funksjonen du skal bruke: x**2

Hva er startverdien? 0

Hva er sluttverdien? 5
```

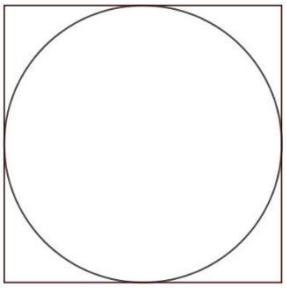


Øvelse 7. Beregning av pi

I denne øvelsen skal vi bruke Python til å beregne en tilnærmet verdi for pi. Dette skal vi gjøre på to måter. I den første delen skal vi bruke tilfeldige tall til å beregne pi, mens i den andre delen skal vi bruke rekkeutvikling av arctangens funksjonen. Metoden med tilfeldige tall inneholder ikke mer matematikk enn at den bør være mulig å gjennomføre på ungdomstrinnet. Metoden med arctangens funksjon er enkel å kode, men skal få en forstå matematikken bak, må en kunne litt om trigonometriske funksjoner samt rekkeutvikling

Beregning av pi ved hjelp av Monte Carlo simulering

Dette er en svært interessant måte å beregne pi på og en slik øvelse bør være fullt mulig å få til for elever på ungdomstrinnet. Prinsippet er ganske enkelt. Vi viser dette med en figur.



Dette er figur der radiusen til sirkelen er lik 1. Det medfører at lengden på sidene til kvadratet er 2. Arealet av sirkelen er $\pi r^2 = \pi$ i vårt tilfelle. Arealet av kvadrater er $2 \cdot 2 = 4$. Vi skal nå generere et tilfeldig tallpar (x, y) der både x og y ligger mellom -1 og 1. Ser vi på arealet av figuren så ser vi at det er $\frac{\pi}{4}$ sjanse for at dette tallet ligger innenfor sirkelen. Det vi skal gjøre i programmet er at vi skal genere mange slike tallpar og deretter avgjøre om de ligger innenfor sirkelen eller ikke. Vi beregner så andelen av tallpar som ligger innenfor sirkelen. Teoretisk sett vil andelen være $\frac{\pi}{4}$. La oss illustrere med et eksempel. Vi genererer 1000 slike tallpar og det viser seg at 793 ligger innenfor sirkelen. Andelen av tallpar som ligger innenfor sirkelen blir dermed

$$\frac{793}{1000} = 0{,}793$$

Siden den teoretiske andelen er $\frac{\pi}{4}$ kan vi estimere pi til å være

$$\pi = 0{,}793 \cdot 4 = 3{,}168$$

Dette er ikke helt nøyaktig. Vi skal se at om vi velger flere tall, vil resultatet bli mer nøyaktig. Det er ganske enkelt å programmere dette og vi skal nærmere på hvordan vi gjør dette.

Vi starter programmet med å legge inn litt tekst, antall tilfeldige tall vi skal trekke samt at vi skal importere funksjonene vi skal bruke. I dette tilfeller det random funksjonen vi trenger.

```
print("I dette programet skal vi bruke en Monte carlo simulering for å beregne pi.")
print("Vi genererer to tilfeldige tall mellom -1 og 1 og regner ut summen av kvadratene.")
print(" Vi teller opp hvor mange av disse summene som ligger innenfor enhetssirkelen. ")
print("Andelen som ligger innenfor multiplisert med 4 gir et anslag for pi.")

n=int(input("Hvor mange tall vil du generere? "))
import random
```

Vi ser at antall trekninger er laget i variabelen n. Det neste vi skal gjøre er å definere noen variable og legge til en startverdi til disse. De vi har bruk for er det som er skrevet opp under

```
i=1
pii=0
```

Variabelen i er en tellevariabel som vi skal bruke til å holde styr på antall ganger vi går gjennom løkken der vi trekker ut et tallpar. Variabelen pii brukes til å telle opp antall ganger tallparet ligger innenfor enhetssirkelen.

Det neste vi skal gjøre er å lage en løkke som vi gjennomgår n ganger. Vi skal i hver repetisjon generere et tallpar (x,y) og bestemme om det ligger innenfor enhetssirkelen eller ikke. Denne gang bruker vi en *while* løkke, men *for* løkke ville fungert like bra.

```
while i<n+0.1:
    x=random.uniform(-1,1)
    y=random.uniform(-1,1)
    p=x**2+y**2
    if p<=1:
        pii=pii+1
    i=i+1
```

Vi satt i til å være lik 1 i forrige avsnitt. Vi skal derfor la repetisjonen fortsette så lenge i er mindre eller lik n. Det gjør vi ved å la repetisjonene fortsette så lenge i<n+0.1. Vi kunne selvsagt brukt et annet lite tall istedenfor 0.1. Poenget er bare å velge noe slik at vi får med oss at i=n. I hver repetisjon genererer vi to tilfeldige tall mellom -1 og 1. Det gjør vi ved å bruke funksjonen

```
random.uniform(-1,1)
```

Vi tilordner det første tilfeldige tallet til variabelen x og det andre til y. Likningen for enhetssirkelen er som vi vet

$$x^2 + y^2 = 1$$

Vi beregner nå hva summen av kvadratene er av våre to tilfeldige tall. Vi kaller resultatet for p. Deretter sjekker vi om p er mindre eller lik 1. Dersom det er tilfelle øker vi verdien med pii med 1. Til slutt øker vi variabelen i med 1. Når løkken er gjennomgått n ganger, vil antall ganger vi har fått et tall der summen av kvadratene er mindre eller lik 1 være lagret i pii.

Vi avslutter programmet med å skrive ut den tilnærmede verdien til pi

```
print("Tilnærmet verdi for pi er:",pii/n*4)
```

Husk at du må gange anslaget du fikk i sted med 4. Komplett program vil se ut som vist under. (Jeg har tatt bort innledningsteksten som forklarer hva programmet gjør). Programmet finner du også i ovelse7a.py

```
n=int(input("Hvor mange tall vil du generere? "))
import random
i=1
pii=0
while i<n+0.1:
    x=random.uniform(-1,1)
```

```
    y=random.uniform(-1,1)
    p=x**2+y**2
    if p<1:
       pii=pii+1
    i=i+1

print()
print("Tilnærmet verdi for pi er:",pii/n*4)
print(
```

Resultatet vil se ut som vist under. Her er det gjennomført 1 000 000 trekninger. Hver oppmerksom på at du kan få et litt annet resultatet for pi siden dette er basert på tilfeldige tall.

```
Hvor mange tall vil du generere? 1000000

Tilnærmet verdi for pi er: 3.141244
```

Utvide programmet med figur

Programmet vi nettopp har laget kan også utvides ved at vi tegner inn punktene som ligger innenfor enhetssirkelen i et diagram. Dette er en nyttig øvelse for å visualisere det vi har programmert samtidig som det også gir nyttig trening i å jobbe med diagram. Du finner hele programmet i filen som heter ovelse7b.py. Vi skal ta utgangspunkt i forrige program og utvide det. Vi starter opp med å importere det vi har bruk for av diagrammer og funksjoner samt å lese inn antall tilfeldige tall vi skal generere.

```
n=int(input("Hvor mange tall vil du generere? "))
import random
import numpy as np
import matplotlib.pyplot as plt
```

Det neste vi skal gjøre er å generere de tilfeldige tallene. I tillegg til det vi gjorde i forrige avsnitt skal vi legge inn x og y verdiene i en liste. Dette må vi gjøre for å få plottet diagrammet etterpå. Vi kan modifisere koden vi brukte i sted som vist under

```
i=1
pii=0
X=[]
Y=[]

while i<n+0.1
   x=random.uniform(-1,1)
   y=random.uniform(-1,1)
   p=x**2+y**2
   if p<=1:
      X.append(x)
      Y.append(y)
      pii=pii+1
   i=i+1
```

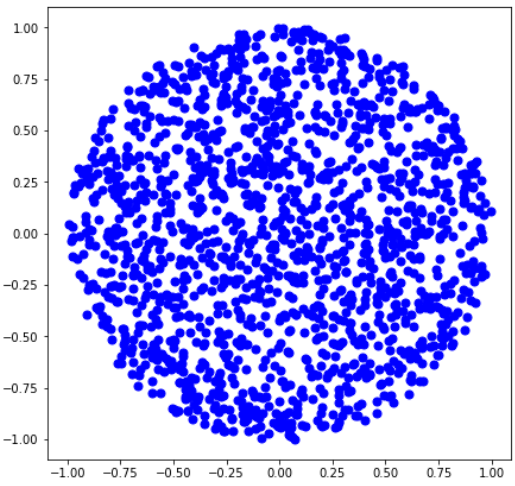
Vi skal se nærmere på det som er nytt fra koden i forrige avsnitt. Vi ser at vi definerer listene X og Y innledningsvis og setter dem tomme i starten ved hjelp av kommandoene X=[] og Y[]. Selve løkken er lik det vi gjorde i sted, bortsett fra at vi for hver gang vi får et tallpar som ligger innenfor enhetssirkelen legger vi til et nytt element i listene X og Y. Det gjør vi med *append* kommandoen. Når vi har gått gjennom løkken det antall ganger vi har lest inn innledningsvis vil alle tallparene som ligger innenfor enhetssirkelen ligge i listene X og Y. Når dette er gjort er vi klare til å lage plottet. Det gjør vi med følgende kommandoer

```
plt.figure(figsize=(7,7))
plt.plot(X,Y,'bo', markersize=7)
plt.show()
```

Kommandoen plt.figure bestemmer størrelsen på figuren. Kommandoen plt.plot tegner opp plottet vårt. Den bruker elementene i listen og tegner opp punktene i koordinatsystemet. Vi har i valgt 'bo' som format på hvordan verdiene skal plottes. Bokstaven b står for farge som i dette tilfellet er blå. Bokstaven o betyr at vi skal makere punktene som sirkler med størrelse 7. Prøv deg frem med hvilken størrelse på diagram og punkter som du synes er passende. Vi bruker til slutt kommandoen *plt.show()* for å vise diagrammet. Dette er som tidligere nevnt ikke nødvendig om du bruker Spyder.

Eksempel på hvordan resultatet blir med 2000 tilfeldige tall er vist under

```
Hvor mange tall vil du generere? 2000
```




```
Tilnærmet verdi for pi er: 3.158
```

Vi ser at punktene ligger innenfor sirkelen. Velger vi flere punkter vil vi få dekket sirkelen enda bedre. Prøvd deg frem selv med flere punkter.

Vi kan utvide programmet enda litt mer ved å tegne inn enhetssirkelen. Vi skal på hvordan dette kan gjøres. I tillegg til koden vi allerede har laget må vi legge inn kommandoer som tegner selve sirkelen. Vi skal gjøre dette i to operasjoner. Vi skal tegne inn en graf for positive y verdier og graf for negative y verdier. Mer presist skal vi tegne inn grafene til funksjonene

$$f(x) = \sqrt{1 - x^2}$$

$$g(x) = -\sqrt{1 - x^2}$$

Koden for hvordan vi gjør dette er vist under.

```
xg = np.linspace(-1,1,1001)
yg=[]
ym=[]
for i in range(0,1001):
    yg.append(sqrt(1-(xg[i])**2))
    ym.append(-(sqrt(1-(xg[i])**2)))
print()
plt.plot(xg,yg,'r', linewidth=5)
plt.plot(xg,ym,'r', linewidth=5)
plt.show()
```

La oss se nærmere på hva som er gjort. Siden vi skal ha to grafer må vi bruke tre lister. En for x-verdiene som vi kaller for xg og en for hver av funksjonene som jeg har kalt for yg og ym. Kommandoen

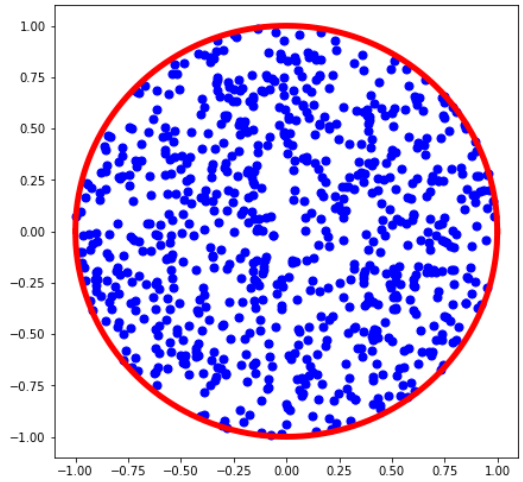
```
xg=np.linspace(-1,1,1001)
```

tilordner x verdiene vi skal bruke i en liste. Vi skal starte på -1 og gå opp til 1. Vi skal til sammen ha 1001 punkter. Vi gjennomgår deretter løkken og beregner for hver x verdi som har betegnelsen xg[i] den tilhørende y verdien for begge funksjonene. Dette lagres i liste yg og ym. Når det er gjort er vi klar til å tegne grafene. Det gjøres med kommandoene

```
plt.plot(xg,yg,'r-', linewidth=5)
plt.plot(xg,ym,'r-', linewidth=5)
plt.show()
```

Prinsippet er det samme som i sted, bortsett fra at vi denne ganger velger rød som farge og strek mellom punktene. Dette er synliggjort med kommandoen 'r-'. Til slutt bestemmer vi tykkelsen på linjene våre. Jeg har satt det til 5, men prøv deg frem med det du synes er passende. Grafen vil nå se slik ut med 2000 punkter.

```
Hvor mange tall vil du generere? 1000
```




```
Tilnærmet verdi for pi er: 3.156
```

Beregne pi ved hjelp av rekkeutvikling

Vi kan vise at funksjonen $\arctan(x)$ kan rekkeutvikles som følgende rekke

$$\arctan(x) = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \frac{1}{9}x^9 - \frac{1}{11}x^{11} \ldots$$

Samtidig vet vi at $\arctan(1) = \frac{\pi}{4}$. Utnytter vi dette ser vi at π kan skrives som følgende rekke

$$\pi = 4 \cdot (1 - \tfrac{1}{3} + \tfrac{1}{5} - \tfrac{1}{7} + \tfrac{1}{9} - \tfrac{1}{11} + \cdots)$$

Det er svært enkelt å programmere dette. La oss se på hvordan vi kan gjøre det i Python.
Programmet som ordner dette er vist under. Vi skal her forklare hvordan du går frem for å lage dette.

```
print("Dette programmet beregner en tilnærmet verdi for ved å rekkeutvikle arctan funksjonen")
print("Vi kan vise at pi/4=1-1/3+1/5-1/7+1/9+..... Vi skal beregne pi ved å velge n ledd i rekken.")
n=int(input("Skriv inn antall ledd i rekken: "))

from math import pi
p=0

for i in range(0,n):
   p=p+(-1)**(i)**1/(2*i+1)

print()
print("En tilnærmet verdi for pi er:    ",p*4)
print("Verdi for pi med 15 desimaler er: ",pi)
print("forskjellen er:                  ",(abs(p*4-pi)))
```

Først legger vi en blank linje og et par linjer med tekst som forklarer hva programmet gjør. Deretter legger vi antall ledd som rekken skal ha ved hjelp av input funksjonen. Resultatet lagres i variabelen n. Vi legger også en startverdi for tilnærmingen til pi. Vi kaller denne variabelen for p og setter den lik 0. Vi må også importere pi fra math biblioteket. Deretter kom det som er interessant. Vi skal bruke en *for* løkke til å beregne rekken. (Vi kunne også brukt en *while* løkke til dette.) Vi beregner først selve rekken og til slutt multipliserer vi med 4 for å finne en tilnærming til pi.

Vi bruker som nevnt en *for* løkke til selve beregningen. Vi setter range til (0,n). Det betyr at i starter med 0 og slutter med n-1 og løkken repeteres da n ganger. En formel for det n'te leddet i rekken kan vi matematisk skrive som dette

Ledd n: $\frac{(-1)^n}{2n+1}$

Det er viktig at du forsikrer deg om dette stemmer. Vi prøver med de 3 første n verdiene, som altså er 0, 1 og 2 hos oss.

$$n = 0 \rightarrow \frac{(-1)^0}{2 \cdot 0 + 1} = 1$$

$$n = 1 \rightarrow \frac{(-1)^1}{2 \cdot 1 + 1} = -\frac{1}{3}$$

$$n = 2 \rightarrow \frac{(-1)^2}{2 \cdot 2 + 1} = \frac{1}{5}$$

Vi ser at dette stemmer

I Python kan vi uttrykke dette som

```
p=p+(-1)**(i)/(2*i+1)
```

Legg merke til at vi hele tiden legger til det neste leddet til verdien til p. Vi starter som kjent med at p=0. Vi ser at selve uttrykket for rekken samsvarer med det vi fant på forrige side. Første gang løkken gås gjennom tar den 0 og legger til verdien for n=0 som altså er 1. Neste gang tar den summen av dette som er 1 og trekker fra 1/3. Ny sum er 2/3. I neste steg legges det til 1/5. Slik fortsetter programmet til vi har gått gjennom alle n stegene.

Når det er gjort skriver vi ut resultatet. Husk at vi må gange p med 4. Vi skriver også ut "eksakte" verdien for pi. Jeg skriver eksakt i anførselstegn da dette ikke er en eksakt verdi, men den som ligger innebygget i programmet. Til slutt skriver vi ut avviket. Du kan prøve deg litt frem med ulike valg av n for å se hvor mange ledd du må ha for å få en akseptabel tilnærming. Prøver du med n=5000 vil du få ut et resultat som vist under.

```
Dette programmet beregner en tilnærmet verdi for ved å rekkeutvikle arctan funksjonen
Vi kan vise at pi/4=1-1/3+1/5-1/7+1/9+..... Vi skal beregne pi ved å velge n ledd i rekken.

Skriv inn antall ledd i rekken: 5000

En tilnærmet verdi for pi er:        3.141392653591791
Verdi for pi med 15 desimaler er:  3.141592653589793
Forskjellen er:                      0.00019999999800202062
```

Programmet for beregning av pi med rekke finner du i filen ovelse7c.py.

Øvelse 8. Kast med en terning

Python er et glimrende verktøy for å gjennomføre terningsimuleringer. I denne øvelsen skal vi se nærmere på utfallet når vi kaster en terning et gitt antall ganger. Vi skal bruke både løkker og *if* setninger for å få dette til. Til slutt skal vi lage en grafisk fremstilling som viser relativ frekvens for utfallene. Selve øvelsen er ganske enkel å programmere, men for å få det til må vi introdusere noen nye kommandoer. Vi beskriver de etter hvert som vi får bruk for dem. Python har en funksjon som genererer tilfeldige tall for oss. Funksjonen som genererer tilfeldige heltall heter randint. Den genererer et tilfeldig heltall mellom to spesifiserte grenser.

Det må påpekes at dette programmet kan gjøres kortere og mer elegant ved å bruke lister. Dette er imidlertid et hakk mer krevende programmeringsteknisk, så i dette programmet bruker vi enkle metoder. I neste øvelse der vi skal se på kast med to terninger skal vi bruke lister og gjøre programmet mer elegant.

La oss starte på programmet vårt der vi skal simulere kast med en terning. Vi stater med å skrive inn koden som er vist under

```
print()
print("Dette programmet kaster en terning n ganger og regner deretter ut frekvens.")
n = int(input("Antall kast:"))

a1=0
a2=0
a3=0
a4=0
a5=0
a6=0
```

La oss forklare litt hva disse kommandoene gjør. Vi starter med å skrive inn en liten tekst som forklarer hva programmet gjør. Deretter legger vi inn antall kast som vi skal gjennomføre med input funksjonen. Vi lagrer resultatet i variabelen n. Vi skal telle opp antall ganger vi får de ulike utfallene på terningen. Vi skal lagre antall enere i variabelen a1, antall toere i a2 osv. Vi legger inn 0 som verdi på disse variablene fra starten av. Vi er nå klare til å gå videre med programmet vårt. De neste kommandoene vi skal utføre er

```
from random import randint
for i in range(1,n+1):
    t=randint(1, 6)
    if t == 1:
       a1=a1+1
    if t == 2:
       a2=a2+1:
    if t == 3:
       a3=a3+1
    if t == 4:
       a4=a4+1
    if t == 5:
       a5=a5+1
```

```
if t == 6:
   a6=a6+1
```

Vi skal se nærmere på hva disse kommandoene utfører. Det første vi legger merke til er at vi må importere funksjonen randint fra pakken random. Dette er vanlig i Python. Svært mange av funksjonene vi skal bruke må importeres slik vi gjør her. Deretter kommer en *for* setning. Vi skal utføre løkken n ganger. Vi starter med i=1 og teller oss opp til i=n. Legg merke til at vi må skrive n+1 inne i parentesen etter range siden for setningen teller til angitt verdi og ikke til og med angitte verdien. Hadde vi skrevet n ville programmet stoppet på n-1. Vi kunne også brukt en *while* setning, men det er smak og behag hva en foretrekker. Det som står med innrykk etter *for* setningen er det som utføres hver gang programmet går gjennom løkken. Vi starter med å generere et tilfeldig heltall mellom 1 og 6 ved hjelp av kommandoen *randint*. Resultatet lagres i variabelen t. Deretter kommer 6 *if* setninger der vi sjekker hva utfallet av terningkastet er. Vi forklarer den første av dem. De andre er helt like. Vi starter med å undersøke om utfallet på terningen er en ener ved å sjekke om t=1 ved hjelp av *if* setningen. Dersom det faktisk er tilfelle så øker vi verdien på variabelen a1 med 1 ved å utføre kommandoen

```
a1=a1+1
```

I Python kan en også skrive dette som a1+=1. Om du skriver det på ene eller andre måten har ikke noe stor betydning og er egentlig smak og behag. Programmet fortsetter med å sjekke om utfallet er 2, 3, osv. Når det har gått gjennom alle seks *if* setningene, så starter det på nytt med et nytt kast. Det gjentas n ganger.

Vi har nå lagret antall enere i a1, antall toere i a2 osv. Det neste vi skal gjøre er å skrive ut resultatet. Det gjør vi ved å bruke *print* kommandoen.

```
print("Antall enere      ",a1)
print("Antall toere      ",a2)
print("Antall trere      ",a3)
print("Antall firere     ",a4)
print("Antall femmere ",a5)
print("Antall seksere   ",a6)
```

Kommandoen for alle linjene er den samme. Vi starter med en tekst som forklarer hva vi skal vise som Antall enere i første linjen. Deretter kommer variabelen og den henter da resultatet fra a1. Legg merke til at det er mellomrom mellom teksten og anførselstegnet. Dette er lagt inn slik at tallene skal komme rett under hverandre. Da blir utskriften pen.

Ofte er vi interessert i å vise den relative frekvensen. Det kan vi gjøre enkelt gjøre. Jeg viser første linjen her. De andre blir helt tilsvarende

```
print("Andel enere   ",round(a1/n,4))
```

Denne kommandoen er ganske lik den foregående. Forskjellen er at vi deler a1 på n for å finne relative frekvensen. I tillegg bruker vi *round* funksjonen for å runde av tallet til fire desimaler. Her kan du selvsagt velge så mange desimaler som du selv ønsker.

Programmet er nå ferdig og vi har et program som simulerer n kast med terning.

Det kan ofte være hensiktmessig å vise resultatet i et stolpediagram og vi skal se på hvordan det kan gjøres. Igjen viser vi kommandoene først og forklarer prinsippet bak dem

```
import matplotlib.pyplot as plt
import numpy as np

height = [a1/n, a2/n, a3/n, a4/n, a5/n, a6/n]
bars = ['1', '2', '3', '4', '5' , '6']
ypos = np.arange(len(bars))
plt.ylim(0,0.25)
plt.bar(ypos, height)
plt.xticks(ypos, bars)
plt.show()
```

- Vi starter med å importere matplotlib modulen. Det gjør vi i første linjen. Vi må også importere numpy som er en modul for å gjennomføre matematiske beregninger.
- height angir høyden på stolpene våre i diagrammet. Dette er en liste.
- bars angir navnet på stolpene og dette vil vises på x-aksen.
- Linjen som starter med ypos lager en tabell med x verdier basert på listen. Vi trenger denne for å få tegnet diagrammet.
- plt.ylim angir verdiene på y-aksen. I dette tilfellet starter vi på 0 og diagrammet stopper på 0.25.
- plt.bar er en kommando for å tegne stolpene. Her kan vi også angi flere parametre som f. eks tykkelse på stolpene.
- plt.xticks er kommandoen for å skrive navnet på stolpene på x-aksen.
- plt.show() tegner til slutt hele diagrammet. Denne trenger vi ikke ha med om vi bruker spyder, men kjører du programmet fra terminalen. Det er uansett greit å ta med denne.

Da er hele programmet ferdig og klar til å testes ut. Du finner programmet i ovelse8.py.

```
print()
print("Dette programmet kaster en terning n ganger og regner deretter ut frekvens.")
n = int(input("Antall kast: "))
a1=0
a2=0
a3=0
a4=0
a5=0
a6=0
from random import randint
for i in range(1,n+1):
    t=randint(1, 6)
    if t == 1:
        a1=a1+1
    if t == 2:
        a2=a2+1
    if t == 3:
        a3=a3+1
    if t == 4:
        a4=a4+1
    if t == 5:
        a5=a5+1
    if t == 6:
        a6=a6+1
```

```
print('')
print("Utfall")
print()

print("Antall enere      ",a1)
print("Antall toere      ",a2)
print("Antall treere     ",a3)
print("Antall firere     ",a4)
print("Antall femmere ",a5)
print("Antall seksere    ",a6)

print('')
print("Sum:                ", a1+a2+a3+a4+a5+a6)

print('')
print("Relativ frekvens")
print('')

print("Andel enere     ",round(a1/n,4))
print("Andel toere     ",round(a2/n,4))
print("Andel trere     ",round(a3/n,4))
print("Andel firere    ",round(a4/n,4))
print("Andel femmere ",round(a5/n,4))
print("Andel seksere    ",round(a6/n,4))

import matplotlib.pyplot as plt
import numpy as np

height = [a1/n, a2/n, a3/n, a4/n, a5/n, a6/n]
bars = ('1', '2', '3', '4', '5' , '6')
ypos = np.arange(len(bars))
plt.ylim(0,0.25)
plt.bar(ypos, height)
plt.xticks(ypos, bars)
plt.show()
```

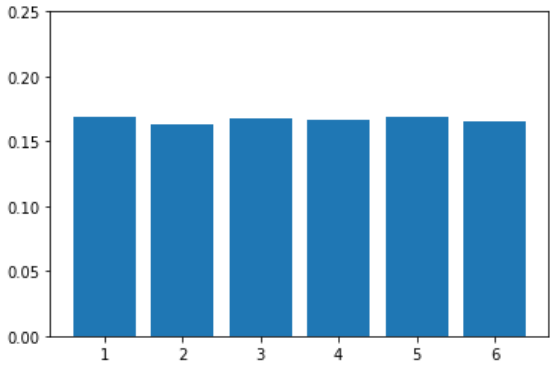
Dersom vi velger 15000 kast vil vi få ut følgende resultat. NB. Husk at dette er tilfeldige tall, så du får ut andre verdier enn det jeg har fått ut.

```
Dette programmet kaster en terning n ganger og regner deretter ut frekvens.

Antall kast: 15000

Utfall

Antall enere      2524
Antall toere      2441
Antall trere      2511
Antall firere     2499
Antall femmere  2540
Antall seksere    2485
```

Sum: 15000

Relativ frekvens

Andel enere 0.1683
Andel toere 0.1627
Andel trere 0.1674
Andel firere 0.1666
Andel femmere 0.1693
Andel seksere 0.1657



Øvelse 9. Lage pene utskrifter

I en tidligere øvelse antydet jeg at vi senere skulle se på hvordan vi kan tilpasse utskriftene slik at de blir litt penere. For å få det til må vi bruke *print* funksjonen på en litt annen måte enn det vi tidligere er vant til. Dette er uvant i starten, men når en har vendt seg til det er det en ganske elegant måte å presentere utskriftene på. Vi viser dette med et lite eksempel. La oss si vi skal skrive ut 3 tall og kvadratroten til disse tallene. Det kan vi gjøre med følgende kommandoer.

```
print("Kvadratroten til 4 er", sqrt(4))
print("Kvadratroten til 50 er", sqrt(50))
print("Kvadratroten til 200 er", sqrt(200))
```

Når en kjører dette vil en få følgende resultat

```
Kvadratroten til 4 er 2.0
Kvadratroten til 50 er 7.0710678118654755
Kvadratroten til 200 er 14.142135623730951
```

Dette er ikke særlig pent. Tallene er ikke under hverandre og det er ulikt antall desimaler i tallene. Dette kan vi som sagt gjøre mer elegant. Vi tar første setningen først der vi ser på 4 og roten av 4. Kommandoen vi skal bruke er vist under.

```
print("Kvadratroten til %3d er %6.3f" %(4,sqrt(4)))
```

Denne krever litt forklaring. Det som skal skrives ut er det som står mellom anførselstegnene. Programmet begynner forfra med å skrive ut helt til det kommer til % tegnet. Når det står et % tegn betyr det at vi skal hente en verdi fra det som står i parentesen. Når vi kommer til første % tegn hentes første verdi. Koden 3d som står umiddelbart etterpå betyr at vi skal sette av 3 plasser til tallet og d betyr at dette er et heltall. Deretter fortsetter programmet med å skrive tekst frem til neste % tegn. Da hentes neste verdi fra parentesen som i vårt tilfelle er roten til 4. Her er koden 5.3f. Det betyr at det settes av 6 plasser til tall, vi skal ha 3 desimaler og f'en forteller at det er et desimaltall vi skal ha inn. Teksten avsluttes med anførselstegn. Deretter skrives det et nytt % tegn og inne i parentesen skrives verdiene vi skal ha med i teksten. Det kan være tall som her eller variabler. Skriver vi alle tre setningene på dette formatet får vi

```
print("Kvadratroten til %3d er %6.3f" %(4,sqrt(4)))
print("Kvadratroten til %3d er %6.3f" %(50,sqrt(50)))
print("Kvadratroten til %3d er %6.3f" %(200,sqrt(200)))
```

Resultatet blir som vist under

```
Kvadratroten til   4 er  2.000
Kvadratroten til  50 er  7.071
Kvadratroten til 200 er 14.142
```

Vi ser at nå kommer tallene rett under hverandre. De er også høyrejustert. Vi kunne venstrejustert dem. Da må vi sette inn – tegnet rett etter % tegnet. Vi kan også ha en tekststreng som vi setter inn med % tegnet. Da skriver vi inn %15s som angir at vi setter av 15 plasser og at det er en tekststreng.

Ved å bruke denne måten å skrive ut på så har vi mye bedre kontroll for å få ryddige og pene utskrifter. Det er som nevnt uvant i starten, men etter hvert så vil du komme inn i denne måten å tenke på. Du finner filen som ovelse9.py

Øvelse 10. Reise til månen ved å brette papir

I denne øvelsen skal vi se på hvor mange du må brette et A4 ark før det blir så tykt at det når helt til månen. Du vil kanskje innvende at et ark ikke kan brettes mer enn 7 ganger og det er for så vidt riktig, men vi kan tenke at vi klipper det i to og legger oppå hverandre istedenfor å brette. Vi skal lage et program der den som kjører programmet skal gjette hvor mange brettinger som trengs for å komme til månen.

Før vi går i gang med å programmere må vi se litt på hva hvor tykt arket blir etter et gitt antall brettinger. Et A4 ark er 0,1 mm tykt. Avstanden til månen er 384 000 km i gjennomsnitt. Funksjonen

$$f(x) = 0{,}1 \cdot 2^n$$

gir tykkelsen på arket etter n brett. I programmet har vi behov for å vite svaret. Det kan enkelt regnes ut ved å sette opp følgende likning

$$0{,}1 \cdot 2^n = 384\,000\,000\,000$$

Husk at funksjonen gir svaret i mm så vi må også regne om avstanden til månen til millimeter. Likningen over løses ved først å dele begge sider på 0,1 og deretter ta logaritmen på begge sider. Det gir oss

$$\ln(2^n) = \ln(3\,840\,000\,000\,000)$$

$$n \cdot \ln(2) = \ln(3\,840\,000\,000\,000)$$

$$n = \frac{\ln(3\,840\,000\,000\,000)}{\ln(2)} = 41{,}8$$

Det tar med andre ord 42 brettinger før vi når helt til månen. Da kommer vi også litt forbi månen.

Da er vi klare til å ta fatt på å lage selve programmet. Vi starter med å legge inn litt tekst som forteller hva programmet skal gjøre.

```
print("Dette programmet regner ut hvor tykt et ark blir når du bretter")
print("det ett visst antall ganger. Et A4 ark er 0.1 mm tykt og du skal")
print("prøve å gjette hvor mange ganger arket må brettes før det når til månen.")
print("Avstanden til månen er ca. 384 000 km")


gjenta="j"
```

Variabelen gjenta er en variabel vi skal bruke for å avgjøre om vi skal gjenta forsøket eller ikke. Vi setter den til "j" innledningsvis.

Det neste vi skal gjøre er å lage oss en funksjon som beregner tykkelsen på arket etter n brettinger. Vi vet at arket i utgangspunktet er 0,1 mm tykt og at det fordobles for hver gang vi bretter det. Vi definerer derfor funksjonen $f(x)$ slik

```
def f(x):
  verdi=2**x*0.1
  return verdi
```

Denne funksjonen beregner tykkelsen og returner verdien som f(x). Det neste vi skal gjøre er å lese inn antall brettinger og avgjøre om dette er nok til å nå månen. Vi skal også gi bruker mulighet til å

gjenta forsøket så mange ganger hun ønsker det. Vi skal derfor lage en *while* løkke der vi gjør dette. Vi starter med følgende kommandoer:

```
while gjenta=="j":
  n=int(input("Hvor mange brettinger tror du trengs for å komme til månen? ")
  print()
```

Vi legger en tom *print* setning for å få litt luft i teksten. Vi skal så legge inn fire *if* setninger der vi skriver ut tykkelsen til arket. Grunnen til at vi bruker fire *if* setninger er at vi skal gi svaret i mm der det er mest hensiktsmessig, meter der det er mest hensiktsmessig og kilometer når det er mest fornuftig. Dette skal stå som en del av det som skal utføres i *while* løkken, altså med samme innrykk som det over.

```
if n<=14:
  print("Med",n,"brettinger er vi kommet %d mm opp" %(f(n)))
if n<=23 and n>=15:
  print("Med",n,"brettinger er vi kommet %d meter opp" %(f(n)/1000))
if n>=24 and n<=1000:
  z=int(f(n)/1000/1000)
  y='{:,}'.format(z).replace(',', ' ')
  print("Med",n,"brettinger er vi kommet",y,"kilometer opp")
if n>=1000:
  print("Nå er arket så tykt at det datamaskinen ikke lenger klarer å beregne hvor tykt det
  er. Prøv med et mindre tall.")
```

Merk. Siste print setning skal stå på en linje i programmet. Den er skrevet på to linjer siden det ikke er plass på en linje i dokumentet.

Om antall brettinger er mindre enn 14 vil tykkelsen være under en meter og da oppgir vi svaret i millimeter. Vi ser at vi ikke trenger å gjøre noe spesielt med funksjonen da. Dersom n er større eller lik 15 og samtidig mindre eller lik 23 så vil svaret være mellom 1 meter og 1 kilometer. Det mest hensiktsmessige da er å oppgi svaret i meter. Vi ser vi må dele funksjonsverdien på 1000 for å få det til. Hvis n er større enn 24 så ønsker vi å oppgi svaret i km. Vi må derfor dele på 1 000 000. Du legger kanskje merke til setningen som står under *if* setningen

```
y='{:,}'.format(z).replace(',', ' ')
```

Dette er en funksjon som lager tusenskilletegn. I Python er kommandoen i utgangspunktet

```
y='{:,}'.format(z)
```

Denne kommandoen setter komme mellom hvert tredje tall. Det betyr at om z=1234354234, så er y=1,234,354,234. I Norge er det mer vanlig å bruke mellomrom fremfor komma og når vi setter på replace(',', ' ') etter format setningen så erstatter Python kommaet med mellomrom. Vi mellomlagrer svaret i variabelen z før vi bruker setningen over til å lage tusen skilletegn. Her kan vi få ganske store tall i svaret og da er det greit å bruke tusen skilletegn.

Vi setter også en begrensing på 1000 brett for at maskinen skal skrive ut svaret. Det gjøres ganske enkelt fordi at dette blir et gigantisk stort tall. Øker vi n enda mer kommer vi ganske snart til et punkt der datamaskinen ikke klarer å beregne svaret.

I tillegg til selve svaret som vi akkurat har skrevet ut kan en gjerne legge på noen kommentarer vedrørende hva svaret blir til den som prøver å gjette på hvor mange brett som trengs. Her kan en selvsagt bruke andre grenser enn det jeg har gjort.

```
if n<=37:
   print("Det er ennå langt igjen til månen. Prøv igjen")
if n<=41 and n>=38:
   print("Du nærmer deg månen!.")
if n==42:
   print("Du har brettet arket akkurat nok ganger til å nå månen.")
if n>=43 and n<=51:
   print("Du er langt forbi månen.")
if n>=52 and n<=68:
   print("Du er langt forbi månen og faktisk forbi solen også.")
if n>=69 and n<=1000:
    print("Du er langt forbi månen og faktisk også forbi nærmeste stjerne.")

gjenta=input("Vil du gjette på nytt? (j/n): ")
print("--------------------")
```

Jeg har lagt inn setningene over som skrives ut umiddelbart etter hvor tykt arket er. Om vi har gjettet på 37 eller mindre vises setningen med at det er langt igjen. Fra 38 til 41 brett så får en beskjed om at det nærmer seg, mens på 42 så får en beskjed om at det er akkurat nok brettinger. Om vi bretter for mange ganger, hvilket det er stor sjanse for så har jeg lagt inn en kommentar med at en enten er forbi månen, forbi solen og til sist forbi nærmeste stjerne som er 4,2 lysår unna. Til slutt har jeg lagt inn spørsmål om vi skal gjenta forsøket. Svarer vi ja ved å trykke j så vil *while* løkken repeteres. I motsatt fall avsluttes programmet. Helt til slutt har jeg lagt inn en stiplet linje for å skille forsøkene fra hverandre.

Om du prøver programmet med 45 brettinger får du ut resultatet som vist under.

```
Dette programmet regner ut hvor tykt et ark blir når du bretter
det ett visst antall ganger. Et A4 ark er 0.1 mm tykt og du skal
prøve å gjette hvor mange ganger arket må brettes før det når til månen.
Avstanden til månen er ca. 384 000 km

Hvor mange brettinger tror du trengs for å komme til månen? 45

Med 45 brettinger er vi kommet 3 518 437 kilometer opp
Du er langt forbi månen.

Vil du gjette på nytt? (j/n): n
--------------------
```

Hele programmet finner du i filen ovelse10.py

Øvelse 11. Kast med to terninger

I denne øvelsen skal vi simulere kast med to terninger der vi beregner summen av terningene. Vi skal gjennomføre mange kast og antall kast leser vi inn i programmet i starten av programmet. Dette programmet kunne vi laget ved å bruke samme teknikk som programmet der vi simulerte kast med 1 terning. Eksempel på hvordan det kan gjøres er vist i ovelse11a.py. Det blir litt skriving om vi gjør det på den måten, så i denne øvelsen skal vi prøve å gjøre programmet litt kortere og mer elegant. Vi skal gå stegvis gjennom hvordan programmet kan konstrueres.

I programmet skal vi legge inn antall ganger vi får de ulike utfallene i en liste. Denne listen kaller vi for a. Vi trenger også en liste der vi legger inn utfallene. Den kaller vi for b. Til slutt skal vi lage en liste med de teoretiske verdiene som vi kaller for t. Du kan starte programmet med å skrive inn følgende kommandoer

```
print("Dette programmet kaster to terning n ganger og regner ut summen av øynene.")
n = int(input("Antall kast: "))

a=[0,0,0,0,0,0,0,0,0,0,0]
b=[2,3,4,5,6,7,8,9,10,11,12]
t=[1/36,2/36,3/36,4/36,5/36,6/36,5/36,4/36,3/36,2/36,1/36]

from random import randint
```

Vi starter med å legge inn litt tekst samt at vi leser inn antall kast vi skal gjennomføre. Deretter lager vi listene. Vi kunne nok brukt en løkke for å lage disse listene, men det går i grunnen raskere å gjøre det manuelt når det ikke er flere utfall enn det er. Verdiene i liste t representerer sannsynligheten for de ulike utfallene.

Sum 2 = $\frac{1}{36}$ Sum 3= $\frac{2}{36}$ Sum 4= $\frac{3}{36}$ osv.

Du kan bruke figuren på under til å forsikre deg om at de sannsynlighetene vi har funnet er de riktige.

11	21	31	41	51	61
12	22	32	42	52	62
13	23	33	43	53	63
14	24	34	44	54	64
15	25	35	45	55	65
16	26	36	46	56	66

Figuren viser alle utfall med to terninger der den ene er rød og den andre er blå. Vi ser at det bare er 1 av 36 utfall som gir sum på 2 og det er to enere. Tilsvarende ser vi at det er to muligheter av 36 for å få sum på 3, at den rød ender med 1 og blå med 2 eller motsatt.

Det siste vi gjør her er å importere funksjonen randint fra random pakken.

Det neste vi skal gjøre er å kaste terningene n ganger og deretter telle opp utfallene. Til dette skal vi bruke en *while* løkke for å gjennomføre n trekninger. Koden vil skal bruke er vist under.

```
i = 0
while i < n:
   u=randint(1,6)
   v=randint(1,6)
   s=u+v
   for j in range(0,11):
      if s==b[j]:
         a[j]=a[j]+1
i=i+1
```

Her er det behov for en litt grundig forklaring da koden kan virke nokså gresk ved første øyekast. Vi starter med å definere en tellevariabel i til å være 0. Deretter kommer *while* løkken som ber oss gjenta det som står i løkken så lenge i er mindre enn n som er kastene våre. Vi definerer deretter to tilfeldige tall med *randint* funksjonen. Vi kaller dem u og v og summen av tallene for s. Vi husker fra kast med en terning at vi måtte ha 6 *if* setninger for å sjekke utfallet av terningene. Skulle vi gjort dt samme her, måtte vi ha brukt 11 *if* setninger. Vi skal derfor heller lage en løkke. Denne gang bruker en *for* setning. Vi skal la j gå gjennom heltallene fra 0 til 10. Hver gang vi er i for løkken sjekker vi verdien av s mot det som ligger på plass j i listen b. Dersom det stemmer legger vi til 1 på tilsvarende plass i liste a. La oss illustrere med et eksempel. La oss si at vi har fått 4 i sum på terningene våre. Da er s lik 4. Når vi går inn for løkken er j=0 første gang og vi sjekker om s er lik det som ligger i b[0] som er 2. Det er ikke tilfelle, så da starter på igjen med for løkken. Da er j blitt til 1. Vi sjekker om s er lik det som ligger i b[1]. Det er heller ikke tilfelle siden s er lik 4 og b[1] er lik 3. Da hopper vi ett steg videre. Denne gang er j lik 2. Vi ser at b[2] er lik 4 som jo er det samme som s. Disse er med andre ord lik og vi utfører det som står i inntrykket under *if* setningen. Der står det at vi skal legge til 1 i posisjon 2 i liste a i vårt tilfelle. Forsikre deg om at du henger med på dette før du går videre. Etter at vi har gjennomgått alle n kastene vil utfallene av terningkastene ligge i liste a. Det kan ofte være lurt å skrive ut liste a når du har skrevet inn kodene for å se at det stemmer. Prøv med et lite antall kast og se om det du får er rimelig. Liste a skriver du ut med kommandoen

```
print(a)
```

Da skrives hele listen ut.

Det neste vi skal gjøre er å skrive ut en oversikt over utfallene våre. Når vi kastet en terning lage vi 6 *print* setninger for å få skrevet ut utfallene. Denne gang skal vi bruke en løkke slik at vi slipper å skrive inn 11 setninger.

```
print()
print("Frekvens")
print('')
for j in range(0,11):
    print("Sum %2d : %7d" % (j+2,a[j]) )
```

De tre første setningene er for å legge inn litt luft og teksten frekvens. Deretter kom en for setning som gjennomgås 11 ganger. Den starter på 0 og slutter på 10 som tilsvarer posisjonene i listene våre. For hver gang vi gjennomgår løkken skriver vi ut en setning. Vi skal se litt nærmere på hva den faktisk gjør. Vi starter med programmet skriver Sum. Deretter kommer et prosenttegn. Vi ser at det er satt av 2 posisjoner til tallet og at det er helttall, siden det står en d. Tallet hentes fra første verdi i

parentesen til slutt, altså j+2. Vi må bruke j+1 siden vi starter med j=0 og første utfallet er 2. Deretter kommer et : som er en tekst for neste % tegn kommer. Her skal vi legge inn utfallet. Vi setter av 7 posisjoner til dette siden vi skal kunne gjennomføre store trekninger. Også her har vi et heltall som vi symboliserer med en d. Verdien henter vi fra listen a og vi henter det som står i posisjon j. Ved første gjennomgang er det a[0] som er antall ganger vi har fått sum på 2. Kjører du programmet med det vi har lagt inn så langt bør du få en utskrift som vist under om du velger 10000 kast

```
Dette programmet kaster to terninger n ganger og regner ut summen av
øynene.

Antall kast: 10000

Frekvens
Sum  2 :     291
Sum  3 :     548
Sum  4 :     851
Sum  5 :    1045
Sum  6 :    1403
Sum  7 :    1652
Sum  8 :    1444
Sum  9 :    1151
Sum 10 :     802
Sum 11 :     536
Sum 12 :     277
```

Husk at vi generert tilfeldige tall. Du vil derfor få andre verdier enn det jeg har fått, men layouten på utskriften bør være lik.

Vi skal også lage en utskrift som viser den relative frekvensen samt et stolpediagram som viser resultatet visuelt. Vi bruker samme teknikk som over for å skrive ut den relative frekvensen. I tillegg til relativ frekvens skal vi også ta med den teoretiske verdien samt avviket mellom vårt forsøk og teoretisk verdi. Kommandoene som er vist under hjelper oss med dette.

```
print('')
print("Utfall    Relativ  Teori    Avvik")
for j in range(0,11):
    print("Sum %2d : %8.4f %8.4f %8.4f " % (j+2, a[j]/n, t[j], abs(a[j]/n-t[j])  ))
print('')
```

Det første vi gjør er å lage en liten overskrift til tabellen vår som indikerer hva som er utfall, relativ frekvens, teoretisk verdi og avvik. Deretter bruk vi en for løkke etter samme prinsipp som i sted til å skrive ut resultatet. Her skal vi skrive ut fire variabler. Den første er som i sted summen av øyne. Den er j+2. Deretter skal vi skrive ut relativ frekvens. Dette er et desimaltall. Jeg har satt av 8 posisjoner til tallet og jeg ønsker å ha med 4 desimaler. Derfor skrives det som %8.4f. Verdien henter jeg fra andre verdi i parentesen som er utfallet delt på antall observasjoner. Den tredje variabelen er den teoretiske verdien. Også her bruker vi 8 plasser og fire desimaler. Verdien henter vi fra listen t. Til slutt skal vi ta med avviket. Avviket skriver vi som absoluttverdien til differansen mellom relativ frekvens og teoretisk verdi. Når du har føyd til denne koden til programmet bør denne delen av utskriften se ut som viset under.

Utfall	Relativ	Teori	Avvik
Sum 2 :	0.0291	0.0278	0.0013
Sum 3 :	0.0548	0.0556	0.0008
Sum 4 :	0.0851	0.0833	0.0018
Sum 5 :	0.1045	0.1111	0.0066
Sum 6 :	0.1403	0.1389	0.0014
Sum 7 :	0.1652	0.1667	0.0015
Sum 8 :	0.1444	0.1389	0.0055
Sum 9 :	0.1151	0.1111	0.0040
Sum 10 :	0.0802	0.0833	0.0031
Sum 11 :	0.0536	0.0556	0.0020
Sum 12 :	0.0277	0.0278	0.0001

Vi ser at vi får en pen utskrift når vi gjør det på denne måten. De kan selvsagt bruke flere plasser til hvert tall eller et annet antall desimaler. Her er det bare å prøve seg med og finne det du synes passer best.

Det siste vi skal gjøre er å tegne diagrammet. Det gjør vi ved å sette inn følgende kommandoer

```
import numpy as np
import matplotlib.pyplot as plt

h=[]
for j in range(0,11):
    h.append(a[j]/n)

y = np.arange(len(b))

plt.ylim(0,0.25)
plt.bar(y, h)
plt.xticks(y, b)
plt.show()
```

Vi ser nærmere på hva disse kommandoene gjør. Vi starter med å importere det vi trenger for å lage diagrammet. Deretter definerer vi en tom liste h. I denne listen skal vi legge inn den relative frekvensen. Vi bruker en løkke til dette. Vi går gjennom den 11 ganger og for hvert steg tar vi og deler frekvensen som ligger i liste a og deler på n. Kommandoen *np.arange(len(b))* kalkulerer først lengden på liste b. Den er som vi vet på 11 elementer. Deretter lager den en ny liste som vi kaller for y som inneholder 11 elementer som starter på 0 og øker med 1 helt til vi kommer på 10. På godt norsk betyr det at listen ser ut som vist under.

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

Vi trenger denne listen for å finne posisjonen vi skal tegne opp stolpene. Kommandoen *plt.ylim(0,0.25*) angir minste og største y verdi i diagrammet. Kommandoen plt.bar(y,b) tegner opp stolpene. I posisjon 0 som altså finnes i y[o] tegnes en stolpe som like høy som verdien i b[0]. De andre stolpene tegnes opp på samme måte. Kommandoen *plt.xticks(y, b)* angir navnet på stolpene. Navnet hentes fra liste b. Et diagram med 3000 kast kan se ut som vist på neste side.

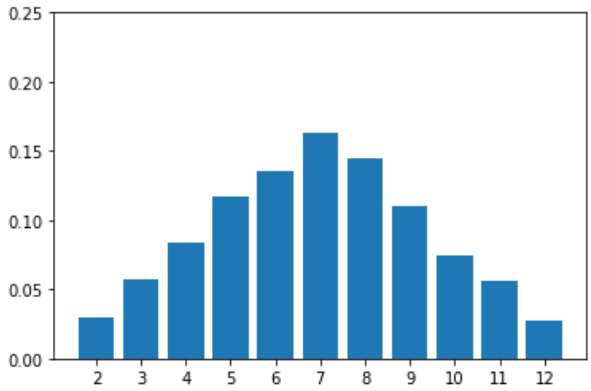
Dette programmet kaster to terninger n ganger og regner ut summen av øynene.

Antall kast: 3000

Frekvens

Sum 2 :   90
Sum 3 : 173
Sum 4 : 252
Sum 5 : 351
Sum 6 : 408
Sum 7 : 488
Sum 8 : 434
Sum 9 : 330
Sum 10 : 225
Sum 11 : 167
Sum 12 :   82

Utfall	Relativ	Teori	Avvik
Sum 2 :	0.0300	0.0278	0.0022
Sum 3 :	0.0577	0.0556	0.0021
Sum 4 :	0.0840	0.0833	0.0007
Sum 5 :	0.1170	0.1111	0.0059
Sum 6 :	0.1360	0.1389	0.0029
Sum 7 :	0.1627	0.1667	0.0040
Sum 8 :	0.1447	0.1389	0.0058
Sum 9 :	0.1100	0.1111	0.0011
Sum 10 :	0.0750	0.0833	0.0083
Sum 11 :	0.0557	0.0556	0.0001
Sum 12 :	0.0273	0.0278	0.0004



Hele koden finner du i filen ovelse11b.py

Øvelse 12. Trekke ut 2 kort blant fire kort der 3 er kløver og et er ruter

I denne øvelsen skal vi se nærmere på en aktivitet der vi har skal trekke ut to kort blant fire kort. De fire kortene er 3 kløvere og en ruter. Spørsmålet nå er om det er størst sjanse for å trekke ut to kløvere eller en kløver og en ruter. Dette skal vi bruke Python til å simulere. Dette er for øvrig en flott aktivitet å bruke i skolen innenfor kombinatorikk og sannsynlighetsregning. En video som viser aktiviteten finner du her.

https://www.youtube.com/watch?v=r8zB0mYBa_8

Nå skal vi konsentrere oss om å programmere simuleringen i Python. Hele øvelsen finner du i filen ovelse12.py. Vi starter med å legge inn litt tekst som forklarer aktiviteten samt antall forsøk

```
print("Dette programmet trekker ut to kort av fire, der tre kort er kløvere og en er ruter.")
print("Programmet regner deretter ut antall ganger vi får kløver-kløver og kløver-ruter")
print("I programmet har vi navngitt kortene med K1, K2, K3 og R7")
print("som står for kløver ess, kløver 2, kløver 3 og ruter 7")
n=int(input("Skriv inn antall trekninger: "))
m=input("Ønsker du å skrive ut alle utfallene? (j/n): ")
```

Vi har i tillegg lagt inn et spørsmål om vi skal skrive ut utfallene eller ikke. Det kan være greit å gjøre for å få visualisert trekningene, men pass på at du bare gjør dette om ikke antall trekninger er for stort. Det neste vi skal gjøre er litt forarbeid før selve trekningen av kortene.

```
from random import *
import numpy as np
import matplotlib.pyplot as plt
kort = ["K1", "K2", "K3", "R7"]
antKK=0
antKR=0
```

Vi må som tidligere importere random samt det som trengs for å tegne diagrammet. I tillegg må lage en liste der de fire kortene er elementene. Jeg har valgt kløver ess, kløver 2 og kløver 3 samt ruter 7 som kort. Det spiller ingen rolle hvilke kort du velger bare det er 3 kløvere og en ruter. Listen har vi bruk for nå vi skal gjennomføre trekningen. I tillegg skal vi bruke variablene antKK og antKR til å telle opp antall ganger vi får kløver-kløver og kløver-ruter. Når dette er gjort er vi klar til å ta fatt på selve trekningen samt å kartlegge hvilket utfall vi har fått. Vi bruker en løkke til dette. Koden som hjelper oss med å utføre dette er vist under.

```
i = 0
while i < n:
    res=sample(kort,k=2)
    res.sort()
    h1=res[0]
    h2=res[1]
    if m=="j":
        print(h1,h2)
    if (h1=="K1" and h2=="K2") or (h1=="K1" and h2=="K3") or (h1=="K2" and h2=="K3"):
        antKK=antKK+1
    else:
```

```
            antKR=antKR+1
        i=i+1
```

Vi skal forklare de ulike stegene. Vi starter med å lage en *while* løkke som gjennomgås n ganger. (Vi kunne også brukt *for* løkke.) Funksjonen *sample(kort,k=2)* er en funksjon som trekker ut to elementer fra listen kort og lagrer det i en ny liste som vi har kalt for res. Når et element er trukket ut, vil det ikke bli trukket ut på nytt med denne kommandoen. Om vi ønsker å trekke ut et annet antall elementer endrer du verdien etter k fra 2 til det du skal trekke ut. Listen res inneholder nå to elementer. Vi sorterer den fra i stigende rekkefølge. Det gjør vi med kommandoen *res.sort*. Vi kaller deretter første element i den sorterte listen for h1 og det andre for h2. Det neste vi sjekker er om vi har bedt om at utfallene skal skrives ut eller ikke. Om vi har valgt ja (ved å skrive j) så vil programmet skrive ut utfallet. Ellers blir det ikke skrevet ut. Neste steg er å sjekke hvilket utfall vi faktisk har fått. Treningsresultatet ligger nå i h1 og h2 og det er disse to vi må sjekke. Vi har tre muligheter for å få kløver-kløver. Det er k1k2, k1k3 og k2k3. Vi sjekker derfor om et av disse vilkårene er oppfylt. Som du ser så skiller vi mellom disse med en *or* kommando. Ser vi nærmere på første uttrykket i *if* setningen (h1=="K1" and h2=="K2") så betyr dette at vi sjekker om h1 er lik K1 samtidig som h2 er lik K2. Er dette oppfylt så øker vi tellevariabelen antKK med 1. På tilsvarende måte sjekker vi de andre utfallene. Siden listen er sortert så må vi ha K1 eller K2 som første element i listen i de situasjonene vi har trukket ut 2 kløvere. Om vi ikke har trukket ut to kløvere, ja da må vi ha trukket ut en kløver og en ruter. Da øker vi antKR med en.

Det som gjenstår da er å skrive ut resultatet og eventuelt lage et diagram. Koden under hjelper oss med det.

```
print()
print("Antall kløver-kløver: %7d (%4.2f%%)" % (antKK, antKK/n*100) )
print("Antall kløver-ruter : %7d (%4.2f%%)" % (antKR, antKR/n*100) )

h = [antKK/n, antKR/n]
bars = ('Kløver-Kløver', 'Kløver-Ruter')
plt.title("Diagram som viser fordeling i prosent. n = {}".format(n))
y = np.arange(len(bars))

plt.ylim(0,0.75)
plt.bar(y,h)
plt.xticks(y,bars)
plt.show()
```

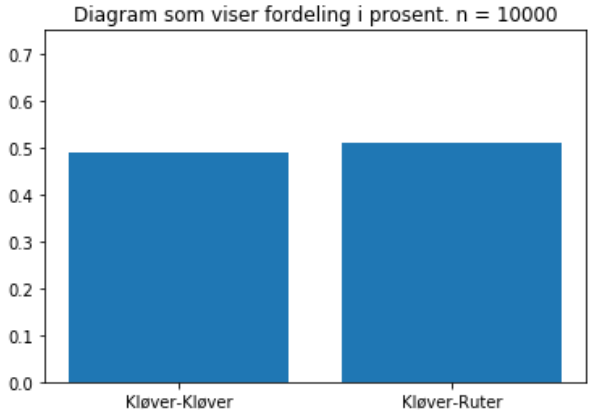
Print setningene skriver ut selve resultatet. Her bruker vi skrivemåten fra øvelse 9. Første tallet er antall ganger med de to utfallene. Dette er et heltall og vi setter av 7 plasser til tallet. Neste tall er relative frekvensen. Her setter vi av 4 plass og vi tar med 2 desimaler. Legg merke til at når vi skal skrive % tegnet som en del av teksten må skrive det som %%. Plotting av diagrammet er helt tilsvarende som i øvelse 10, så ta en titt på det som står der om du trenger forklaring på kodene. Hele resultatet for 10000 kjøringer vi se ut som vist på neste side.

Dette programmet trekker ut to kort av fire, der tre kort er kløvere og en er ruter.
Programmet regner deretter ut antall ganger vi får kløver-kløver og kløver-ruter
I programmet har vi navngitt kortene med K1, K2, K3 og R7
som står for kløver ess, kløver 2, kløver 3 og ruter 7

Skriv inn antall trekninger: 10000

Ønsker du å skrive ut alle utfallene? (j/n): n

Antall kløver-kløver: 4897 (48.97%)
Antall kløver-ruter : 5103 (51.03%)



Øvelse 13. Numerisk integrasjon

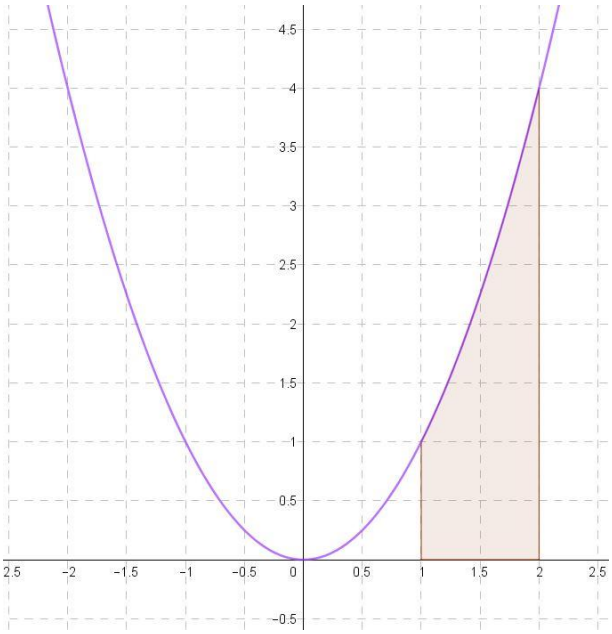
Dette er en øvelse som er fin å gjøre om du er kjent med integralregning. Vi skal i øvelsen se nærmere på hvordan vi kan bruke Python til å beregne bestemte integral numerisk. Verdien av det bestemte integralet

$$\int_a^b f(x)dx$$

er verdien av arealet mellom grafen og x-aksen for område mellom $x = a$ og $x = b$. Integralet

$$\int_1^2 x^2 dx$$

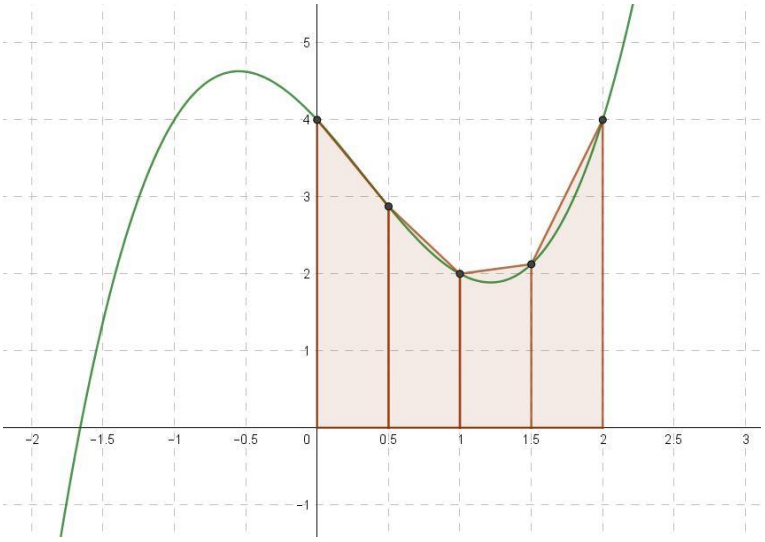
vil tilsvare det fargede området på figuren under



Dette gjelder om funksjonen er positiv i området. Dersom den er negativ får vi også arealet, men med et negativt fortegn. Mange integral kan vi beregne ved å regne ut den antideriverte, men langt i fra alle. Integralet $\int e^{x^2}\, dx$ er ikke mulig å beregne ved å bruke antideriverte. Med andre ord, det er ingen funksjon som har e^{x^2} som sin deriverte. Eneste måten å beregne et slikt integral er å bruke numerisk integrasjon. I denne øvelsen skal vi se nærmere på to måter som vi kan bruke for å beregne bestemte integraler. Vi skal først se på en metode som kalles trapesmetoden som går ut på at vi regner ut arealet til mange trapeser under kurven istedenfor arealet. Den andre er en metode som heter rektangelmetoden. Den går ut på at vi tilnærmer arealet ved å beregne arealet av mange rektangler og så summere dem. Til øvelsen med rektangelmetoden skal vi også tegne opp grafen og tilhørende rektangler. Det gjør vi ikke med trapesmetoden, da det er litt mer plundrete å tegne opp trapeser.

Trapesmetoden

Det er enkelt å lage et Pythonprogram som beregner integral numerisk med trapesmetoden. La oss se figuren under for å få tak i ideen bak metoden.



Her har vi tegnet opp funksjonen $f(x) = x^3 - x^2 - 2x + 4$. Denne skal vi integrere fra 0 til 2. Med andre ord, så skal vi regne ut integralet

$$\int_0^2 (x^3 - x^2 - 2x + 4)dx$$

Regner vi ut dette med antideriverte får vi

$$\int_0^2 (x^3 - x^2 - 2x + 4)dx = \left[\frac{1}{4}x^4 - \frac{1}{3}x^3 - x^2 + 4x\right]_0^2 = 5\frac{1}{3}$$

I figuren har vi også tegnet inn fire trapeser. Vi ser at arealet av disse til sammen ikke er så langt unna arealet under grafen. Regner vi ut summen av trapesene får vi 5,5 som svar. Vi ser at vi har en ganske god tilnærming selv med bare fire trapeser. Øker vi antall trapeser vi vi naturlig nok få en bedre tilnærming. Når vi har programmert dette skal vi se at vi kan velge et stort antall trapeser og få et meget nøyaktig svar.

Da er vi klare til å ta fatt på selve programmet. Vi starter med å skrive inn litt tekst samt at vi leser inn funksjonen vi skal integrere samt grensene.

```
from math import *

print("Dette programmet beregner integralet av funksjonen du skriver inn mellom")
print("de gitte grensene. Metoden som brukes er trapesmetoden.")

formel=input("Skriv inn funksjonen som du skal integrere: ")
a=float(input("Hva er nedre grense? "))
b=float(input("Hva er øvre grense? "))
n=int(input("Hvor mange oppdelinger skal du ha? "))
```

```
def f(x):
    return eval(formel)
```

Vi starter med å importere funksjonene vi skal bruke fra math modulen. Når vi skriver * får vi med alle funksjonene som finnes i modulen. Deretter leser vi inn funksjonen vår og nedre og øvre grense. Vi leser også inn hvor mange trapes vi skal ha. Deretter definerer vi en funksjon f(x) ut av funksjonen vi leste inn. Husk at det vi leser inn er en tekststreng og vi må konvertere den til en funksjon. Det gjøres ved hjelp av *def* kommandoen og *eval* kommandoen.

Vi skal bygge opp dette programmet litt annerledes enn det vi har gjort tidligere. Dette skal vi gjøre for å vise en teknikk som kan være smart å bruke i litt større og mer sammensatte programmer. Vi skal her definere en funksjon som regner ut integralet for oss. Deretter skal vi kalle den opp og skrive ut resultatet. Dette er hensiktsmessig å gjøre om du har behov for å regne ut flere integral i et og samme program. Det programmeringstekniske er likevel svært likt det vi tidligere har gjort. Vi skal starte med å lage en funksjon som beregner arealet av summen av trapesene. I øvelse 6 beskrev vi hvordan vi kan jobbe med funksjoner så gå gjerne tilbake dit om det er behov for å friske opp dette. Vi skal definere en funksjon som vi kaller for integral. Det gjør vi på denne måten

```
h=(b-a)/float(n)

def integral(f,a,b,n):
  sum=0
  for i in range(0,n):
        sum=sum+(f(a+h*i)+f(a+h*(i+1)))*h/2
  return sum
```

Vi starter med å regne ut en verdi vi har kalt h. Vi tar b minus a og deler på n. Da får vi bredden på trapesene vi skal bruke. Det neste vi gjør er å definere funksjonen. Vi kaller den for integral (vi kunne brukt et annet navn også). Den skal ha variablene f, a, b og n. Legg merke til at vi også kan bruke f som variabel. Det neste vi gjør er å lage en løkke som vi gjennomgår n ganger, med andre ord en gang for hvert trapes. Vi starter med å sette sum lik 0. Uttrykket

```
sum=sum+(f(a+h*i)+f(a+h*(i+1)))*h/2
```

beregner arealet av hvert trapes og legger det sammen med summen av det vi allerede har. Formelen for trapes er

$$A = \frac{(a+b)\cdot 2h}{2}$$

La oss se på eksempelet med funksjonen vi hadde i sted og se hva som skjer første gang vi går gjennom løkken. Vi valgte i sted fire oppdelinger og grenser på 0 og 2. Det medfører at $h = 0.5$. Arealet av første trapes blir da

$$A1 = \frac{\big(f(0)+f(0.5)\big)\cdot 0.5}{2} = \frac{\left(4+2\frac{7}{8}\right)\cdot 0.5}{2} = 1{,}71875$$

Neste gang programmet går gjennom løkken vil den regne ut arealet av trapes nummer 2. Det gir oss

$$A2 = \frac{\big(f(0.5)+f(1)\big)\cdot 0.5}{2} = \frac{\left(2\frac{7}{8}+2\right)\cdot 0.5}{2} = 1{,}21875$$

Slik fortsetter programmet til alle trapesene er regnet ut.

Det siste vi skal gjøre er å skrive ut resultatet. Det gjør vi ved hjelp av følgende kommandoer

```
if b>=a:
   ap=integral(f,a,b,n)
   print("Verdien på integralet er %5.3f" %(ap))
else:
   print("Nedre grense er større enn øvre grense. Prøv igjen.")
```

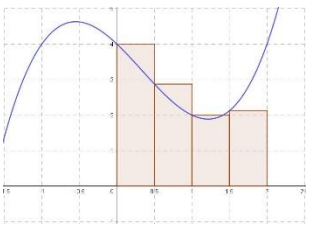
Vi starter med en *if* setning for å avgjøre om øvre grenser er større enn nedre grense. Hvis det er tilfelle kaller vi opp funksjonen vår som heter integral med parameterne våre. Deretter skriver vi det ut. Om øvre grense er mindre enn nedre grense skriver programmet ut en feilmelding til oss. Prøver du programmet med funksjonen vår og 100 trapes får du ut følgende resultatet.

```
Dette programmet beregner integralet av funksjonen du skriver inn
mellom de gitte grensene. Metoden som brukes er trapesmetoden.

Skriv inn funksjonen som du skal integrere: x**3-x**2-2*x+4

Hva er nedre grense? 0

Hva er øvre grense? 2

Hvor mange oppdelinger skal du ha? 100

Verdien på integralet er 5.334
```

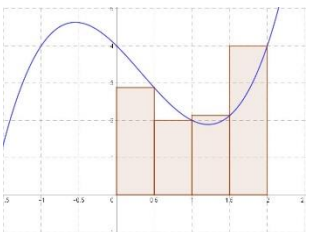
Vi ser vi har en meget god tilnærming til integralet nå. Det er ingenting i veien for å bruke finere oppdeling. Prøvd deg gjerne frem med dette. Du kan gjerne velge et større antall desimaler også for å se hvor lite avviket blir. Hele programmet finner du i filen ovelse13a.py

Rektangelmetoden

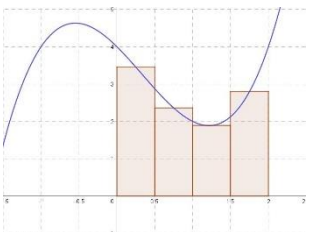
Vi skal nå se nærmere på rektangelmetoden. Det er tre måter en kan bruke den på. Vi kan venstrejustere rektanglene, høyrejustere dem, eller ta utgangspunktet i midtpunktet intervallet. De tre alternativene er skissert under.



Venstrejustert rektangel



Høyrejustert rektangel



Midtjustert rektangel

Midtjusterte rektangler vil normalt gi best tilnærming. Vi ser vi mister litt av arealet på ene siden og får med litt for mye på andre siden for de fleste rektangler. Dette gjør at rektanglene ofte gir en bra tilnærming. Vårt program skal ta utgangspunkt i midtjusterte rektangler. Første delen av programmet

er likt det vi gjorde med trapesmetoden. Du kan derfor ta utgangspunkt i den koden. Vi skal gjøre en liten modifikasjon på den før vi ser på hvordan vi skal tegne grafen og rektanglene.

Fra programmet fra trapesmetoden er det nødvendig å justere på funksjon integral. Vi må modifisere den som vist under

```
def integral(f,a,b,n):
    sum=0
    for i in range(0,n):
        sum=sum+f(a+h*(2*i+1)/2)*h
    return sum
```

Poenget her er at vi finner midten av hvert rektangel og beregner tilhørende funksjonsverdi. Koden kan ved første øyekast virke litt kryptisk. La oss se litt nærmere på den. Midtpunktet til de første rektanglene er

Rektangel 1: a+h/2
Rektangel 2: a+3h/2
Rektangel 3: a+5h/2

Når vi går gjennom løkken og starter med i=0 så ser vi at ved første gang så får vi

f(a+h*(2*0+1)/2*h=f(a+h/2)*h

Vi ser dette stemmer med det vi nettopp har funnet. Neste gang vi går gjennom løkken får vi

f(a+h*(2*1+1)/2*h=f(a+3h/2)*h

Vi ser dette stemmer også med det vi fant. Vi ganger med h som er bredden av rektangelet for å finne arealet.

Prøver vi med 100 oppdelinger vil programmet gi oss følgende resultat

```
Dette programmet beregner integralet av funksjonen du skriver inn mellom de gitte grensene.
Metoden som brukes er rektangelmetoden. Grafen samt rektanglene vises til slutt

Skriv inn funksjonen som du skal integrere: x**3-x**2-2*x+4

Hva er nedre grense? 0

Hva er øvre grense? 2

Hvor mange oppdelinger skal du ha? 100

Verdien på integralet er 5.333
```

Vi ser at resultatet stemmer meget bra. Prøv gjerne med finere oppdeling og flere desimaler også.

Det neste steget er at vi skal tegne opp rektanglene og grafen. Dersom vi velger en veldig fin oppdeling blir det nokså uoversiktlig og vi ser bare en farget flate. Vi skal derfor legge inn et spørsmål om vi skal tegne grafen eller ikke. Vi skal deretter beregne verdiene vi trenger for å tegne grafen. Koden for dette finner du på neste side. Hele programmet finner du for øvrig i filen ovelse13b.py.

```
t=input("Ønsker du å tegne opp grafen. Da bør ikke n være altfor stor: (j/n) ")

if t=="j":
   X = np.linspace(a,b,1001)
   i=1
   Y=[f(a)]
   while i<1000+0.1:
      t=f(a+(b-a)/1000*i)
      Y.append(t)
      i=i+1

   x = np.linspace(a+h/2,b-h/2,n)
   i=1
   y=[f(a+h/2)]
   while i<n:
     t=f(a+h*(2*i+1)/2)
     y.append(t)
     i=i+1
```

Vi starter med å legge inn et spørsmål om vi skal tegne grafen eller ikke. Her skal en svare j eller n. Deretter kommer det en *if* setning. Dersom vi har svart ja på forrige spørsmål ved å trykke j så går vi i gang med å tegne grafen og rektanglene. Vi ser vi har to løkker vi skal gå gjennom. Den ene er for å beregne dataene vi trenger til selve funksjonen og den andre er for å beregne verdiene til rektanglene. Vi ser først på grafen til funksjonen. Vi starter med å definere x verdiene til grafen ved å bruke *np.linspace* kommandoen. Den gir oss 1001 x verdier som er jevnt fordelt mellom a og b. Legg merke til at vi bruker stor X her. Vi definerer deretter en liste Y som skal inneholde tilhørende y verdier. Den første legger vi inn manuelt ved å si at Y=[f(a)]. Dette blir det første elementet i liste Y. Deretter lar vi i være lik 1 og gjennomgår listen 1000 ganger. For hver gang øker vi x verdien med (b-a)/1000. Den tilhørende y verdi blir beregnet. Først kaller vi den for t og deretter legger vi til verdien til liste Y ved hjelp av *append* kommandoen.

Den neste løkken er litt mer komplisert. Her skal vi ha inn x verdien og høyden på rektanglene våre. Vi har valgt en oppdeling på n rektangler som er lest inn. Siden vi skal ha rektangler som tar utgangspunkt i midtverdien lar vi første x verdi være a+h/2. Vi lar sluttverdien være b-h/2 og vi skal ha til sammen n punkter. Vi lagrer dette i listen x. Ta gjerne utgangspunkt 3. gradsfunksjonen vår med fire oppdelinger for å visualisere at dette stemmer. Også her legger vi inn første verdi manuelt. Vi kaller listen med tilhørende y verdier for y. I løkken som vi gjennomgår n-1 ganger beregnes tilhørende y verdi. Prinsippet er det samme som når vi beregnet høyden på rektanglene tidligere i programmet.

Ikke så rent sjelden, vil en oppleve at programmet ikke gir riktig svar første gangen. Det kan ofte lønne seg å skrive ut listene for å se hvilke tall de gir og om de stemmer. Legg gjerne inn print(x) og print(y) i slutten av løkken om du ikke får ut det du ønsker. Da vil det gjerne være lettere å finne feilen.

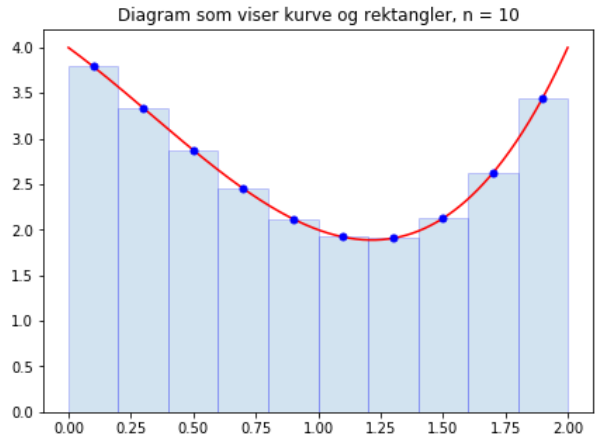
Når vi har laget listene for grafen og rektanglene er vi klar for å tegne diagrammet. Koden under hjelper oss med det. Dette skal også stå som innrykk etter *if* setningen, bortsett fra siste *else* setning.

```
        figure(figsize=(7,5))
        plt.plot(X,Y,'r')
        plt.plot(x,y,'bo',markersize=5)
        plt.bar(x,y,width=(b-a)/n,alpha=0.2,align='center',edgecolor='b')
        plt.title('Diagram som viser kurve og rektangler, n = {}'.format(n))
        plt.show()

    input("Trykk en tast for å avslutte")
```

Merk. Siste *input* setning er ikke med i det som skal utføres i *if* setningen. Vi starter med å definere størrelsen på figuren. Bredde er først og deretter høyden. Her kan du selvsagt velge hvilken størrelse du vil. Vi plotter deretter selve funksjonen vår. Vi bruker bare navnet på listene som variabel. Vi velger 'r-' slik at vi grafen blir en rød linje. Vi plotter deretter midtpunktet til hvert rektangel. Dette skal vises som blå sirkler. Vi velger størrelse 5, men her kan du velge hva du synes passer best. Neste setningen tegner opp rektanglene. Vi bruker da listene x og y som vi laget i sted. Vi setter bredden på rektanglene ved hjelp av width, alpha kommandoen lar oss sette hvor sterk farge det skal være på rektanglene, align kan vi velge left, right eller center. Vi velger center siden de skal være sentrert om x verdiene. Kommandoen plt.title lar oss velge en tittel til figuren.

Velger vi 10 oppdelinger på 3. gradsfunksjonen vår får vi

```
Dette programmet beregner integralet av funksjonen du skriver inn mellom de gitte grensene.
Metoden som brukes er rektangelmetoden. Grafen samt rektanglene vises til slutt

Skriv inn funksjonen som du skal integrere: x**3-x**2-2*x+4

Hva er nedre grense? 0

Hva er øvre grense? 2

Hvor mange oppdelinger skal du ha? 10

Verdien på integralet er 5.320

Ønsker du å tegne opp grafen. Da bør ikke n være altfor stor: (j/n) j
```



Øvelse 14. Simulering av Lottotrekningen

Mange av oss har levert inn en lottokupong for deretter å vente på den magiske telefonen fra Norsk Tipping på lørdagskvelden. Dessverre så uteblir telefonen for de aller fleste av oss. I denne øvelsen skal vi lage et program som simulerer lottotrekningen. Vi skal spille rekken vår et gitt antall ganger. Vi skal først lage et program der vi lager en rekke og deretter spiller den n ganger. Deretter skal vi lage en kupong med ti tilfeldige rekker som vi også spiller n ganger. Når vi skal lage programmet for å spille en rekke skal vi bruke ting vi allerede har jobbet med, men når vi utvider til 10 rekker skal vi innføre noen nye funksjoner.

Sjansen for å vinne i Lotto

Lotto går ut på at det trekkes ut 7 tall av 34 tall. Dette er hovedtallene. I tillegg trekkes det ut ett tilleggstall. Dersom vi klarer å treffe minst 4 av de uttrukne tallene vinner vi en premie. Premiene er som følger:

1. premie: 7 rette
2. premie: 6 rette pluss tilleggstall
3. premie: 6 rette
4. premie: 5 rette
5. premie: 4 rette

I simuleringen vi skal gjøre skal vi også sammenlikne vår simulering med den teoretiske sannsynligheten for å vinne de forskjellige premiene. Det totale antall rekker vi kan spille i Lotto er $\binom{34}{7} = 5\,379\,616$ kombinasjoner. Dersom vi skal vinne førstepremien må vi treffe alle 7 tallene og sjansen for 7 rette blir derfor bare 1 delt på 5 379 616. Når vi skal beregne sannsynligheten for de øvrige premiene ser vi på antall kombinasjoner som gir de ulike premiene og deler det på det totale antall kombinasjoner. Da får vi:

1. premie: $\frac{1}{\binom{34}{7}} = \frac{1}{5\,379\,616}$
2. premie: $\frac{\binom{7}{6}\cdot\binom{1}{1}\cdot\binom{26}{0}}{\binom{34}{7}} = \frac{7}{5\,379\,616}$
3. premie: $\frac{\binom{7}{6}\cdot\binom{27}{1}}{\binom{34}{7}} = \frac{189}{5\,379\,616}$
4. premie: $\frac{\binom{7}{5}\cdot\binom{27}{2}}{\binom{34}{7}} = \frac{7371}{5\,379\,616}$
5. premie: $\frac{\binom{7}{4}\cdot\binom{27}{3}}{\binom{34}{7}} = \frac{102\,375}{5\,379\,616}$

Simulering av lottotrekningen

Vi skal se nærmere på hvordan vi kan bruke Python til å lage en simulering av Lotto trekningen. Vi starter med å lage programmet der vi spiller en rekke n ganger. Dette programmet kan gjøres i mange steg eller vi kan lage det kortere med bruk av løkker. I filen ovelse14a.py finner du et program som er stegvis bygget opp. Vi skal i denne øvelsen bruke løkker i noe større grad. Dette gjør

programmet kortere og mer elegant, men også litt vanskeligere å lese. Programmet slik vi skal lage det finner du i filen ovelse14b.py

Vi skal først legge inn litt tekst som forklarer hva programmet gjør, samt at vi leser inn antall trekninger.

```
print("Dette programmet spiller en lottorekke n uker på rad. Maskinen setter opp rekken ")
print("for deg og gjennomfører deretter det antall trekninger du ber om. Deretter teller ")
print("programmet opp hvor mange premier du vinner.")
n = int(input("Skriv inn antall trekninger: "))
from random import *
```

Vi importerer også funksjonene fra random som vi trenger for å lage tilfeldige tall. Det neste vi skal gjøre er å lage en liste som inneholder tallene fra 1 til 34. Det kan gjøres på to måter. Den ene er ganske enkelt å skrive inn listen som vist under

```
lotto=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34]
```

Dette er ikke særlig elegant. Vi kan også lage listen ved å bruke en løkke. Følgende setninger lager listen for oss

```
lotto=[]
for i in range(1,35):
   lotto.append(i)
```

Vi starter med å definere listen lotto som en tom liste. Deretter går vi gjennom løkken 34 ganger og for hver gang legger vi til et element som har verdien i. Da vil du få generert samme liste. Et interessant spørsmål er hva som er raskest og mest praktisk. I dette tilfelle går det nok raskere å skrive inn de 34 tallene enn å lage en løkke. Hadde vi derimot hatt 1000 elementer, ville det utvilsomt gått mye raskere med å bruke løkke. Det er jo også mer elegant med en løkke. Vi bruker derfor det videre i programmet vårt. Det neste vi skal gjøre er å trekke ut rekken som er vår rekke. Det gjøres med følgende kommandoer

```
min=sample(lotto, k=7)
min.sort()
print()
print("Min rekke: ",min)

ant0=0
ant5=0
ant4=0
ant3=0
ant2=0
ant1=0
```

Kommandoen *sample* trekker, i vårt tilfelle, ut 7 tall fra listen lotto og definerer resultatet som en ny liste som jeg har kalt min. Kommandoen *min.sort()* sorterer listen fra minste tall til største tall. Til slutt skriver vi ut rekken vår. Vi skal bruke variablene ant0 til ant5 til å telle opp antall premier vi vinner. Disse må vi sette lik 0 innledningsvis og det kan vi gjøre like etter at vi har generert rekken vår. Da skal vi ta fatt på selve trekningen og opptelling av antall rette. Kommandoene under hjelper oss med dette.

```
for i in range(0,n):
    trekk=sample(lotto, k=8)
    rette=0
    til1=0

    for m in range(0,7):
        if min[m] in trekk[:-1]:
            rette=rette+1

    if trekk[7] in min:
        til1=1
```

Vi starter med å lage en løkke som vi gjennomgår n ganger som tilsvarer det antall trekninger vi skal gjøre. Det neste vi gjør er å trekke ut 8 tall fra listen lotto. De 7 første tallene er hovedtallene og det siste skal være tilleggstallet. Vi definerer også variablene rette og til1 som vi begge setter lik 0. Disse skal brukes til å telle opp antall riktige og sjekke om vi har tilleggstallet. Det neste vi skal gjøre er å lage en løkke der vi sjekker om hvert enkelt tall i vår rekke er med i vinnerrekken. Løkken starter med å sette m=0 og sjekker dermed det første tallet i vår rekke som er lagret i min[0]. I *if* setningen like under sjekker vi om dette tallet er med i listen trekk som altså utgjør vinnertallene. Legg merke til at vi skriver [:-1] rett etter trekk. Dette gjør fordi vi bare ønsker å sjekke de 7 første tallene i listen og ikke det siste som utgjør tilleggstallet. Dersom tallet vi sjekker finnes i vinnerrekken øker vi rette med 1. Når vi har gått gjennom løkken 7 ganger og sjekket alle tallene våre får vi antall rette vi har fått. Den neste *if* setningen sjekker om tilleggstallet som vi finner i trekk[7] er med blant tallene våre. Om det er det settes til1 til 1. Ellers er den 0 slik vi satte den innledningsvis.

Det neste vi skal gjøre er å sjekke hvor mange rette vi har og legge dette inn i tellevariabelen for premiene. Kommandoene under gjør dette for oss. Husk at disse skal inn som en del av kommandoene som utføres i for løkken med variabelen i.

```
if rette<4:
  ant0=ant0+1
if rette==4:
  ant5=ant5+1
if rette==5:
  ant4=ant4+1
if rette==6 and til1==0:
  ant3=ant3+1
if rette==6 and til1==1:
  ant2=ant2+1
if rette==7:
  ant1=ant1+1
```

Disse linjene er ganske selvforklarende. Først sjekker vi om rette er mindre enn 4. Dersom det er tilfelle øker vi ant0 med 1. Dette er variabelen som teller opp antall ganger vi ikke får premie. Legg merke til at vi for 2. og 3. premie må sjekke både antall rette og hva tilleggstallet er.

Da er programmet i grunnen ferdig og det som gjenstår er å skrive ut resultatet. Vi skal først skrive ut selve resultatet og deretter den relative frekvensen og sammenlikne det med teoretiske verdien. Kommandoene under skriver ut selve resultatet.

```
print("Antall premier: ")
print()
print("%-13s %7d" %("Ingen premie:",ant0))
print("%-13s %7d" %("5. premie:",ant5))
print("%-13s %7d" %("4. premie:",ant4))
print("%-13s %7d" %("3. premie:",ant3))
print("%-13s %7d" %("2. premie:",ant2))
print("%-13s %7d" %("1. premie:",ant1))
```

Vi ser vi setter av 13 plasser til teksten. At vi bruker %-13s indikerer at vi skal venstre justere denne variabelen. Vi setter av 7 plasser til selve resultatet. Siden dette er heltall setter vi %7d.

Vi skal også skrive ut den relative frekvensen og sammenlikne den med teoretiske verdiene. Til dette bruker vi kommandoene

```
print("Tabellen under viser relative frekvensen, teoretisk verdi og avvik.")
print()

e0=5269673/5379616
e5=102375/5379616
e4=7371/5379616
e3=189/5379616
e2=7/5379616
e1=1/5379616

print("%-13s %10s %10s %10s" %("Premie:","Andel","Teori","Avvik"))
print("%-13s %10.7f %10.7f %10.7f" %("Ingen premie:",ant0/n,e0,abs(ant0/n-e0)))
print("%-13s %10.7f %10.7f %10.7f" %("5. premie:",ant5/n,e5,abs(ant5/n-e5)))
print("%-13s %10.7f %10.7f %10.7f" %("4. premie:",ant4/n,e4,abs(ant4/n-e4)))
print("%-13s %10.7f %10.7f %10.7f" %("3. premie:",ant3/n,e3,abs(ant3/n-e3)))
print("%-13s %10.7f %10.7f %10.7f" %("2. premie:",ant2/n,e2,abs(ant2/n-e2)))
print("%-13s %10.7f %10.7f %10.7f" %("1. premie:",ant1/n,e1,abs(ant1/n-e1)))
```

Vi starter med å definere variablene e0 til e5. Her legger vi inn de teoretiske verdiene. Vi gjør dette for at ikke *print* setningene skal bli altfor uoversiktlige. Første *print* setning etter dette er overskrift til tabellen. Deretter kommer tekst som indikerer premiene, deretter følger relative frekvensen. Legg merke til at vi deler ant0 til ant5 på n. Neste vi skal skrive ut er teoretiske verdien og til slitt kommer absoluttverdien til differansen. Jeg har satt av 10 plasser til hvert tall og 7 desimaler. Her kan du selv velge hva du synes er hensiktsmessig.

Når vi gjennomfører en kjøring får du opp et resultat som vist på neste side. Her har vi valgt 1 000 000 trekninger. Vi har med andre ord spilt rekken vår nesten hver lørdag i nesten 20 000 år.

```
Dette programmet spiller en lottorekke n uker på rad. Maskinen setter opp rekken
for deg og gjennomfører deretter det antall trekninger du ber om. Deretter teller
programmet opp hvor mange premier du vinner.

Skriv inn antall trekninger: 1000000

Min rekke:  [3, 5, 9, 12, 20, 22, 33]

Antall premier:

Ingen premie:  979563
5. premie:      19026
4. premie:       1375
3. premie:         35
2. premie:          1
1. premie:          0

Tabellen under viser relative frekvensen, teoretisk verdi og avvik.

Premie:              Andel        Teori        Avvik
Ingen premie:  0.9795630  0.9795630  0.0000000
5. premie:     0.0190260  0.0190302  0.0000042
4. premie:     0.0013750  0.0013702  0.0000048
3. premie:     0.0000350  0.0000351  0.0000001
2. premie:     0.0000010  0.0000013  0.0000003
1. premie:     0.0000000  0.0000002  0.0000002
```

Spille en 10 rekkers kupong

Når vi først skal spille lotto velger de fleste av oss å kjøpe en kupong med 10 rekker som spilles i alt fra 1 uke til 10 uker. Vi skal nå utvide programmet slik at kan spille en 10 rekkers kupong i så mange uker som vi ønsker. Vi skal ta utgangspunkt i det foregående programmet og modifisere det. Hele koden for programmet finner du i filen ovelse14c.py.

For å kunne lage 10 rekker er det enkleste å bruke nøstede lister. Vi skal se på et enkelt eksempel på hvordan en nøstet liste ser ut. Vi skal kalle listen vår for A. Vi skal ta utgangspunkt i to lister som vi kaller for t1=[2,4,7,9] og t2=[3,6,8,9] som vi skal sette sammen til en matrise A. Vi starter med å definere A som en tom liste. Deretter bruker vi *append* funksjonen.

```
A=[]
t1=[3,6,8,9]
t2=[2,4,7,9]
A.append(t1)
A.append(t2)
```

Skriver vi ut A vil vi få følgende resultat

```
[[3, 6, 8, 9], [2, 4, 7, 9]]
```

Det som er litt interessant nå er at vi kan hente ut enkelt elementer fra listen eller hele linjer. Kommandoene

```
print(A[1][2])
```

skriver ut elementet fra andre listen og henter ut element 3 derfra som i vårt tilfelle er 7. Kommandoen

```
print(A[0])
```

skriver ut første listen, altså [3,6,8,9].

Ønsker vi å skrive ut begge listene på en oversiktlig og fin form kan vi gjøre det slik

```
print(A[0])
print(A[1])
```

som gir oss resultatet

```
[3, 6, 8, 9]
[2, 4, 7, 9]
```

Dette er egenskaper som er nyttige og som vi skal utnytte når vi skal be maskinen lage 10 enkeltrekker til oss. På side 56 genererte vi en enkelt rekke. Denne koden skal vi justere slik at den lager 10 rekker til oss. Koden som gjør dette er vist under.

```
ti=[]
for i in range(0,10):
    min=random.sample(lotto, k=7)
    min.sort()
    ti.append(min)
print()
print("Mine rekker: ")
print()
for i in range(0,10):
   print(ti[i])
```

Vi starter med å definere en tom liste som vi kaller for ti. Vi lager deretter en løkke som vi gjennomgår 10 ganger. Det første vi gjør i hver repetisjon er å genere en rekke som vi kaller for min. Vi sortere tallene i stigende rekkefølge. Det er strengt tatt ikke nødvendig, men det ser litt penere ut da. Vi legger deretter til listen min i listen ti. Til slutt skriver vi ut alle ti rekkene. Vi bruker en liten repetisjonsløkke til dette der vi lar i gå fra 0 til 9 og vi skriver ut linje i hver gang.

Det neste vi må gjøre er å modifisere koden for hvordan vi sjekker antall rette. Koden som sjekker en rekke er vist øverst på side 57. Det er denne vi nå må endre. På neste side ser du hvordan koden ser ut etter vi har justert den.

```
for i in range(0,n):
    trekk=random.sample(lotto, k=8)
    for l in range(0,10):
        rette=0
        til1=0
        for m in range(0,7):
            if ti[l][m] in trekk[:-1]:
                rette=rette+1

        if trekk[7] in ti[l]:
            til1=1

        if rette<4:
            ant0=ant+1
```

Setningene som sjekker de øvrige premiene er identisk med forrige program og gjentas ikke her.

Vi skal se nærmere på hvordan denne koden fungerer. Vi legger merke til at vi har fått inn en ny *for* setning der vi har 10 repetisjoner. Den første for løkken går vi gjennom n ganger som er antall trekninger vi gjennomfører. For hver trekning trekker vi vinnertallene samt et tilleggstall. Disse lagres som før i listen trekk. Den neste *for* løkken går vi altså gjennom 10 ganger som tilsvarer antall rekker på kupongen vår. I hver repetisjon sjekker vi hvor mange rette vi har på rekkene våre. La oss prøve å eksemplifisere dette. Første gang vi går inn i løkken er l lik 0. Vi går deretter rett inn i ny *for* løkke. Første gang er m lik 0. I *if* setningen som følger sjekker vi i dette tilfelle om ti[0][0] som er første tallet i første rekken er med i trekk, altså blant vinnertallene. Neste gang er m lik 1 og vi sjekker da om ti[0][1] som er det andre tallet i første rekken er blant vinnertallene. Slik fortsetter vi til vi har sjekket alle tallene i første rekken. Når det er gjort øker l med 1 til 1. Første sjekk i det tilfelle blir om ti[1][0] som er første tallet i andre rekke er med blant vinnertallene. Ser du gangen i dette? Det er ofte lurt å prøve å visualisere koden slik jeg har gjort her. Resten av programmet er likt det vi gjorde i sted med ett unntak og det er at vi må dele ant0, ant1,…, ant5 med $n \cdot 10$ når vi skal beregne den relative frekvensen siden vi spiller 10 rekker i hver trekning.

En kjøring der vi spiller ti rekkers kupongen vår 500 000 uker på rad er vist på neste side. Vi ser at det er rimelig bra samsvar mellom vår simulering og de teoretiske verdiene. Det er selvsagt ikke nødvendig å skrive ut rekkene våre, men jeg har valgt å ta det med her for å vise de ti rekkene vi spiller. Vi kunne skrevet ut treningsresultatet også, men med 500 000 kjøringer må maskinen skrive ut 500 000 linjer og det har ikke noe hensikt. Det som eventuelt kunne vært gjort var å legge inn et spørsmål om trekningsresultatet skal skrives ut eller ikke, slik vi gjorde med figuren på integraloppgaven. Når en utvikler selve programmet er ikke det så dumt å gjøre det for lettere å sjekke at programmet faktisk fungerer som det skal.

Dette programmet spiller en ti rekkers lottokupong n uker på rad. Maskinen setter opp rekkene for deg og gjennomfører deretter det antall trekninger du ber om. Deretter teller programmet opp hvor mange premier du vinner.

Skriv inn antall trekninger: 500000

Mine rekker:

[11, 18, 20, 23, 27, 29, 30]
[1, 3, 5, 7, 20, 25, 26]
[2, 6, 13, 15, 18, 26, 32]
[10, 15, 17, 20, 21, 23, 27]
[2, 9, 12, 18, 22, 27, 31]
[2, 4, 9, 17, 23, 26, 32]
[1, 3, 4, 10, 15, 17, 23]
[6, 12, 18, 19, 23, 25, 27]
[2, 3, 15, 18, 20, 30, 33]
[2, 5, 6, 10, 13, 24, 30]

Antall premier:

Ingen premie:	4897676
5. premie:	95292
4. premie:	6845
3. premie:	178
2. premie:	8
1. premie:	1

Tabellen under viser relative frekvensen, teoretisk verdi og avvik.

Premie:	Andel	Teori	Avvik
Ingen premie:	0.9795352	0.9795630	0.0000278
5. premie:	0.0190584	0.0190302	0.0000282
4. premie:	0.0013690	0.0013702	0.0000012
3. premie:	0.0000356	0.0000351	0.0000005
2. premie:	0.0000016	0.0000013	0.0000003

Øvelse 15. Kast med ball. Animasjon av grafen

I denne øvelsen skal vi se nærmere på kurven en ball følger når vi kaster den med en gitt utgangshastighet og med en bestemt vinkel. Fra fysikken kan vi vise at dersom vi ser bort fra luftmotstand vil en ball følge kurven som er beskrevet med funksjonen under

$$f(x) = \frac{-g}{2 \cdot (v_0 \cdot \cos(\varphi))^2} \cdot x^2 + \tan(\varphi) \cdot x + h$$

der g er tyngeakselerasjonen som vi setter til 9,81. Konstanten h er høyden vi kaster ballen fra. Vi antar her at den kastes fra 2 meters høyde. Det vil si at ballen er 2 meter over bakken når ballen forlater armen. Funksjonen kan da skrives som følgende uttrykk

$$f(x) = \frac{-9{,}81}{2 \cdot (v_0 \cdot \cos(\varphi))^2} \cdot x^2 + \tan(\varphi) \cdot x + 2$$

Siden både utgangshastigheten v_0 og vinkel vi kaster ballen med er kjent så ser vi at dette er en andregradsfunksjon.

Vi skal lage et program i Python som ikke bare tegner opp grafen, men den skal også vise kurven med en animasjon. Programmet skal også regne ut hvor ballen lander og når den er på topp og hva høyden blir i det tilfellet. Jeg vil anbefale at du kjører programmet som ligger i filen ovelse15.py slik at du ser hvordan det ferdige programmet skal se ut. For å få animasjonen til å virke må du kjøre kommandoen

```
%matplotlib qt
```

i Spyder consolet før du kjører programmet. Når du har kjørt programmet og ser hvordan virker så kan du lese videre om hvordan du kan konstruere programmet. Vi starter med å importere programmene vi har bruk for samt å lese inn utgangshastighet og vinkel.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from math import *

plt.close('all')

v0=float(input("Hva er utgangshastigheten i kilometer i timen? "))
fi=float(input("Hva er utgangsvinkelen i grader? "))
```

Vi starter med å importere pakkene og funksjonene vi har bruk for. Det som er nytt er at vi importerer funksjonen FuncAnimation som vi skal bruke til selve animasjonen. Vi lukker også tidligere vindu med *plt.close* funksjonen. Vi leser deretter inn utgangshastigheten og vinkelen og tilordner de til variablene v0 og h. Vi leser inn resultatet i kilometer i timen og vinkelen i grader. I formelen så brukes meter per sekund og radianer og vi må derfor regne om variablene. Kommandoene

```
fi=2*pi*fi/360
v0=v0/3.6
```

sørger for omregningene til meter per sekund og radianer. Det neste vi skal gjøre er å beregne når ballen lander. Siden ballen følger en andregradskurve så vil dette være i punktet hvor funksjonen vår er 0. Med andre ord, så skal vi løse andregradslikningen

$$\frac{-9{,}81}{2 \cdot (v_0 \cdot \cos(\varphi))^2} \cdot x^2 + \tan(\varphi) \cdot x + 2 = 0$$

For å få programmet litt oversiktlig så beregner vi først koeffisientene a, b og c i andregradslikningen

$$ax^2 + bx + c = 0$$

Når vi ser nærmere på funksjonsuttrykket ser vi at koeffisientene kan uttrykkes som vist under i Python

```
a=-9.81/(2*(v0*cos(fi))**2)
b=tan(fi)
c=2
```

I øvelse 4 løste vi andreslikningen. Vi bruker samme prinsipp her, bortsett fra at vi ikke trenger å tenke på om den har løsning eller ikke. Vi vet jo at ballen treffer bakken til slutt, så den vil alltid ha løsning. Løsningen av likningen kan uttrykkes ved hjelp av følgende kode

```
null1=(-b-sqrt(b**2-4*a*c))/2/a
null2=(-b+sqrt(b**2-4*a*c))/2/a
topp=(null1+null2)/2
```

En andregradslikning har som vi vet to løsningen. Vi lager programmet slik at vi finner begge. Vi kaller dem for null1 og null2. Vi har også bruk for å finne midtpunktet mellom nullpunktene siden toppunktet ligger her. Vi kaller det for topp og beregner det samtidig. Det neste skrittet nå er å definere en funksjon i Python for $f(x)$. Vi kaller den for $f(x)$ også i Python

```
def f(x):
    return -9.81*x**2/(2*(v0*cos(fi))**2) + x*tan(fi)+2
```

Denne funksjonen har vi bruk for både når vi skal beregne når ballen er på topp og til selve animasjonen. Det neste vi skal gjøre å skrive ut en tekst der vi har beregnet hvor ballen lander. Vi har to nullpunkt og spørsmålet er hvilket av dem som er landingspunktet. Ser vi nærmere på uttrykkene vil vi se at det er null1 som er landingspunktet, da både telleren og nevneren begge er negative, noe som gir et positivt svar. Vi er da klare til å lage en tekst som forteller oss hvor ballen lander og når den er på topp

```
print("Ballen treffer bakken etter %5.2f meter" %(null1))
print("Ballen er på topp etter %5.2f meter. Da er den %5.2f meter over bakken."
%(topp,f(topp)))
```

Merk. Den andre printsetningen må du skrive som en linje i Spyder. Den er imidlertid for lang til at den fikk plass på en linje i Word. Legg merke til at i setning nummer 2 har vi to verdier vi skal skrive ut. Vi skal skrive ut både hvor den er på topp og hvor høyt den er på det høyeste. Vi bruker funksjonen vi nettopp definerte til det.

Nå er vi klare til å ta fatt på å programmere selve animasjonen. Det er mange måter å gjøre dette på. Vi viser en måte nå som er ganske enkelt. Søker du på litt på nettet vil du finne mye stoff om hvordan en kan lage ulike typer animasjoner. Vi skal starte med koden

```
fig, ax = plt.subplots()
xdata = []
ydata = []
animasjon, = plt.plot([], [], 'r.')
```

Setningen fig, ax = plt.subplots() som vi står først må vi ha i starten. Den genererer en figur og akser og gir selve figuren navnet fig og aksene navnet ax. Dette skal vi bruke senere. Vi genererer deretter to tomme lister, en for x verdier og en for y verdier som vi kaller henholdsvis xdata og ydata. Til slutt genererer vi et tomt plot. Det gjøres ved å bruke *plt.plot* på to tomme lister. Vi legger også til at plottet vårt skal bestå av røde prikker. Legg merke til at vi bruker et komma etter variabelen. Det er fordi at kommandoen genererer et tallpar som vi tilordner til animasjon. Her er det et tomt tallpar siden det er det vi har puttet inn.

Det neste vi skal gjøre er å angi verdiene på aksene. Her kan du selvsagt velge selv hvilke verdier du vil bruke, men det er et poeng at hele kastet får plass på figuren. Jeg har i dette eksemplet valgt å la x aksen gå opp til der ballen treffer bakken pluss 10%. Maks verdi på y-aksen setter jeg i utgangspunktet til 20, men dersom toppunktet til ballen er på mer enn 15 meter bruker jeg maksverdien pluss 20%. Dette er for at grafen skal få god plass i figuren. If-else setningen i koden under beregner maks y verdi som vi skal bruke og tilordner den til variabelen t.

```
if f(topp)<15:
   t=20
else:
   t=f(topp)*1.2

ax = plt.axis([0,null1*1.1,0,t])
step=np.linspace(0, null1, 50)
```

Setningen ax=plt.axis angir minimum og maksimumsverdi til aksene. De to første tallene er x verdiene og de to siste er y verdiene. Vi ser at dette samsvarer med det vi har beskrevet over. Variabelen step er en liste der vi genererer 50 x verdier mellom 0 og punktet der ballen lander. Det som skjer etterpå er at vi egentlig genererer 50 plot som er like bortsett fra siste x og y verdi. Der bruker vi de 50 x verdiene og tilhørende y verdi.

Det neste vi skal gjøre nå er å lage en funksjon som hjelper oss med selve animasjonen. Den skal brukes til å legge inn verdier i xdata og ydata.

```
def oppdater(x):
   xdata.append(x)
   ydata.append(f(x))
   animasjon.set_data(xdata, ydata)
   return animasjon,
```

Vi kaller funksjonen for oppdater, men du kan selvsagt bruke det navnet du ønsker. Den starter med å legge inn en ny verdi i listen xdata ved hjelp av *append* kommandoen. Vi gjør det samme med y verdien. Vi kaller opp funksjonen f(x) for å beregne y verdien. Til slutt bruker vi *set_data()* funksjonen til å returnere objektet som skal plottes til variabelen animasjon.

Da gjenstår det bare å utføre selve plottet. Vi skal bruke funksjonen FuncAnimation til det. Under ser du koden. Vi skal forklare litt nærmere hva vi gjør

```
ani = FuncAnimation(fig, oppdater, frames=step, interval=60, blit=True)
plt.show()
```

Første argument er fig som altså er den tomme figuren vi starter med. Deretter kommer selve funksjonen som gjør oppdateringen. Deretter må vi angi hvor mange ganger vi skal kjøre animasjonen og tilhørede x verdi. Vi genererte dette tidligere og la i listen step. Vi setter derfor frames=step. Interval angir tiden mellom hver animasjon i millisekund. Jeg har satt den til 60 millisekund, men her kan de selvsagt velge selv hva du vil bruke. Når vi setter blit til true forteller vi programmet at kun endringene skal tegnes.

Hele koden vil med det se ut som vist under

```
import numpy as np
import matplotlib.pyplot as plt
from math import *
from matplotlib.animation import FuncAnimation

plt.close('all')

v0=float(input("Hva er utgangshastigheten i kilometer i timen? "))
fi=float(input("Hva er utgangsvinkelen i grader? "))

fi=2*pi*fi/360
v0=v0/3.6

a=-9.81/(2*(v0*cos(fi))**2)
b=tan(fi)
c=2

null1=(-b-sqrt(b**2-4*a*c))/2/a
null2=(-b+sqrt(b**2-4*a*c))/2/a
topp=(null1+null2)/2

print()

def f(x):
   return -9.81*x**2/(2*(v0*cos(fi))**2) + x*tan(fi)+2

print("Ballen treffer bakken etter %5.2f meter" %(null1))
print("Ballen er på topp etter %5.2f meter. Da er den %5.2f meter over
bakken." %(topp,f(topp)))

fig, ax = plt.subplots()
xdata = []
ydata = []
animasjon, = plt.plot([], [], 'r.')

if f(topp)<15:
   t=20
```

```
else:
   t=f(topp)*1.2

ax = plt.axis([0,null1*1.1,0,t])
step=np.linspace(0, null1, 50)


def oppdater(x):
   xdata.append(x)
   ydata.append(f(x))
   animasjon.set_data(xdata, ydata)
   return animasjon,

ani = FuncAnimation(fig, oppdater, frames=step, interval=60, blit=True)
plt.show()
```

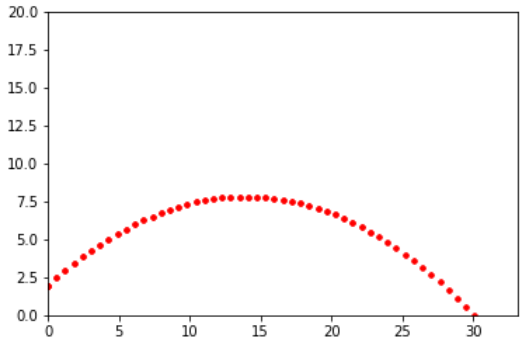
Når du kjører programmet vi du få ut følgende resultat om du velger 60 km/t og 40 grader.

```
Hva er utgangshastigheten i kilometer i timen? 60

Hva er utgangsvinkelen i grader? 40

Ballen treffer bakken etter 30.09 meter
Ballen er på topp etter 13.94 meter. Da er den  7.85 meter over bakken.
```

Og har du husket å kjøre kommandoen

```
%matplotlib qt
```

Vil du i tillegg få ut en figur som dette når animasjonen er ferdig.



Øvelse 16. Monty Hall problemet

Monty Hall problemet stammer fra et amerikansk TV show. Konseptet gikk ut på at en deltaker skulle velge en av tre dører. Bak en av dørene skjulte det seg en Mercedes og bak de to andre skjulte det seg en geit. Deltakeren visste naturligvis ikke hva som skjulte seg bak hvilken dør. Når deltakeren hadde valgt seg en dør, men før døren ble åpnet, så åpnet programlederen en annen dør. Døren som ble åpnet skjulte alltid en geit. Spørsmålet til deltakeren nå er om han fremdeles ønsket innholdet bak døren han opprinnelig hadde valgt, eller om han ønsket å bytte dør. En enkel og intuitiv forklaring på hva som er lurt å gjøre er beskrevet her

https://no.wikipedia.org/wiki/Monty_Hall-problemet

Vi skal lage et program som simulerer hvordan TV showet foregikk. Vi skal etterpå lage en simulering som kjører et valgfritt antall show automatisk og samtidig teller opp antall ganger vi vinner Mercedes ved å bytte dør og ved å stå i ro. Jeg tror det lønner seg å åpne selve programmet først og kjøre det slik at du ser hva vi skal programmere. Du finner programmet som ovelse16.py

Vi er da klare til å ta fatt på selve programmeringen. Vi starter med å importere pakkene vi trenger samt å legge inn litt tekst.

```
from random import *
print()
print("Dette programmet lager en simulering av Mercedes-Geit problemet.")
print()
gjenta="j"
```

Siden deltakeren skal kunne gjennomføre showet så mange hun ønsker så skal vi bruke en *while* løkke videre. Vi definerer derfor variabelen gjenta og legger inn j som startverdi. Vi definerer deretter det som skal gjennomføres i løkken. Vi starter med å generere hva som skal skjule seg bak dørene samt at vi lar deltaker velge en dør.

```
while gjenta=="j":'
   MG=["G","G","M"]
   min=sample(MG, k=3)
   print("Hva skjuler seg bak dørene: ",min)
   n = int(input("Velg en dør ved å skrive inn 1, 2 eller 3: "))
```

Jeg definerer her en liste hvor jeg legger inn to geiter, symbolisert med G og en Mercedes som symboliseres med M. Jeg bruker deretter kommandoen *sample* for å velge ut tre elementer i tilfeldig rekkefølge fra listen. Vi får dermed stokket dem i tilfeldig rekkefølgen. Når en skal lage programmet lønner det seg å skrive ut listen som er stokket for å kunne kontrollere at programmet fungerer som det skal. Når en skal prøve dette ut på noen, tar en bare vekk den *print* setningen. Jeg har tatt den med i filen som ligger ute på nettsiden. Det siste vi gjør i kommandorekken over er å la deltakeren velge dør. Vi lagrer resultatet i variabelen n.

Det neste vi skal gjøre er å programmere inn at programlederen åpner en dør som skjuler en geit. Vi trenger noen *if* setningene for å få det til. Vi skal starte med situasjonen der deltakeren har valgt dør nummer 1.

```
if n==1 and min[0]=="M":
    m=randint(2,3)
if n==1 and min[0]=="G":
   if min[1]=="G":
      m=2
   if min[1]=="M":
     m=3
```

Her trengs det en nærmere forklaring. Vi starter med å sjekke om deltaker har valgt dør 1 og om innholdet i den er en Mercedes. Dersom det er tilfelle så settes variabelen m til 2 eller 3 som velges av maskinen tilfeldig. Variabelen m skal angi hvilken dør som programleder skal åpne med geiten bak. Som vi ser så må han åpne dør 2 eller 3 når Mercedesen er bak dør 1. Den neste *if* setningen sjekker om det er en geit bak dør 1. Dersom det er tilfelle må vi ta en ny sjekk på dør 2 og sjekke om den en geit. Om det er tilfelle skal programlederen åpne den døren og vi setter m til å være 2. Til sist sjekker vi om dør 2 er skjuler en Mercedes. Om den gjør det må programleder åpne dør 3 og vi setter m til å være 3. Merk at vi kunne også brukt *else* setning i stedenfor siste *if* setning.

Vi fortsetter deretter å sjekke om deltaker har valgt dør 2 og hva som skjuler seg bak den.

```
if n==2 and min[1]=="M":
   m=randint(1,2)
   if m==2:
      m=3
if n==2 and min[1]=="G":
   if min[0]=="G":
      m=1
   if min[0]=="M":
      m=3
```

Disse kommandoene er nesten helt lik de vi brukte for å sjekke om deltakeren hadde valgt dør 1. Eneste forskjellen er i første *if* setning. Dersom døren skjuler en Mercedes så skal dør 1 eller 3 åpnes. Vi bruker *randint* til å generere heltallet 1 eller 2. Dersom vi får 2 legger vi inn at dør 3 skal åpnes for å vise geiten. Til slutt tar vi en sjekk på dør 3. Kommandoene her er tilsvarende som for dør 1.

```
if n==3 and min[2]=="M":
    m=randint(1,2)
if n==3 and min[2]=="G":
   if min[0]=="G":
      m=1
   if min[0]=="M":
      m=2
```

Med disse kommandoene har vi nå fått programmert hvilken dør programlederen skal åpne. Vi kan derfor skrive det i programmet. Vi legger også inn et spørsmål om deltakeren vil bytte dør.

```
print()
print("Programlederen åpner nå dør nr", m ,"som skjuler en geit.")
s=input("Vil du bytte dør? (j/n): ")
```

Det kan være lurt å teste programmet nå for å se at det virker så langt. I og med at vi har skrevet ut hva som skjuler seg bak dørene innledningsvis så er det lett å kontrollere.

Det neste vi skal gjøre er å skrive ut hva deltakeren vinner i premie. Det vil naturligvis avhenge av om han bytter dør eller ikke. Det må vi ta hensyn til.

```
if s=="n":
   if min[n-1]=="G":
      premie="Geit"
   if min[n-1]=="M":
      premie="Mercedes"

if s=="j":
    if min[n-1]=="G":
       premie="Mercedes"
   if min[n-1]=="M":
       premie="Geit"


print()
print("Du vinner en",premie)
gjenta=input("Vil du spille en gang til? (j/n): ")
print("--------------------")
```

Vi starter med å sjekke hva premien blir om deltaker ikke bytter dør. Vi må dermed sjekke om døren skjuler en Geit eller en Mercedes. Legg merke til at når deltaker har valgt dør nr. n så finner vi resultatet i posisjon n-1 i listen. Vi lagrer resultatet i variabelen premie. Dersom deltakeren bytter dør ved å skrive j på spørsmålet så vi være litt forsiktig. Dersom vi velger å bytte dør, så vil vi alltid vinne Mercedesen om døren vi først valgte skjuler en geit. I motsatt fall vinner vi en geit. Til slutt skriver vi ut hva premien blir og et spørsmål om vi skal spille showet på nytt.

Når deltakeren ikke ønsker å være med i flere show skal vi gjennomføre en del show automatisk. Vi lar deltakeren selv avgjøre hvor mange show som skal kjøres. Siden vi skriver ut resultatet fra hvert show er det greit å ikke kjøre for mange show og gjerne under 1000 stykker

```
ant=int(input("Hvor mange show vil du kjøre automatisk? "))
antm=0
for i in range(0,ant):
   MG=["G","G","M"]
   min=sample(MG, k=3)
   trekk=randint(0,2)
   if min[trekk]=="G":
      premieb="Mercedes"
      premies="Geit"
      antm=antm+1
   if min[trekk]=="M":
      premieb="Geit"
      premies="Merdedes"

   print("Dørene: ",min,"Valgt dør:",trekk+1," Premie ved bytte: %-10s Premie ved å beholde:
   %-10s" %(premieb,premies))
```

Merk. Siste setningen skal skrives på en linje i Spyder. Her er den delt opp fordi det ikke var plass på en linje. La oss se nærmere på hva denne delen av programmet gjør. Vi starter med å plassere Mercedesen og geitene bak dørene. Vi lar deretter deltakeren velge seg en dør automatisk ved at vi

genererer et tilfeldig heltall mellom 0 og 2. Døren blir da dette tallet pluss 1. Dersom døren skjuler en geit, så vet vi at premien ved å bytte er en Mercedes. Vi lagrer dette i variabelen premieb. Tilsvarende lagrer vi geit i variabelen premies som indikerer hva premien er om vi står i ro. Vi teller også opp antall Mercedeser vi vinner ved å bruke tellevariabelen ant. Vi lar den øke med 1 for hver gang vi vinner en Mercedes ved å bytte. På tilsvarende måte sjekker vi hva premien er om vi starter med en dør med Mercedes bak. Den siste setningen skriver ut resultatet for hvert show. Vi tar med hva som skjuler seg bak dørene, hvilken dør som er valgt og hva vi får i premie ved å bytte og ved å stå i ro. Det aller siste vi gjør er å skrive ut hvor mange ganger vi vinner Mercedes ved å stå i ro og ved å bytte dør. Vi skriver ut både antallet og i prosent.

```
print()
print("Antall Mercedeser ved å bytte:   ",antm,"(%5.2f %%)" %(antm/ant*100))
print("Antall Mercedeser ved å stå i ro:", ant-antm,"(%5.2f %%)" %((ant-antm)/ant*100))
```

Resultatet av en kjøring kan se ut som vist under. Her har vi kjørt showet en gang og valgt å bytte dør og fått fin premie. Deretter er showet kjørt 10 ganger automatisk.

```
Dette programmet lager en simulering av Mercedes-Geit problemet.

Hva skjuler seg bak dørene:  ['G', 'G', 'M']

Velg en dør ved å skrive inn 1, 2 eller 3: 1

Programlederen åpner nå dør nr 2 som skjuler en geit.

Vil du bytte dør? (j/n): j

Du vinner en Mercedes

Vil du spille en gang til? (j/n): n
----------

Hvor mange show vil du kjøre automatisk? 10
Dørene:  ['M', 'G', 'G'] Valgt dør: 1  Premie ved bytte: Geit          Premie ved å beholde: Merdedes
Dørene:  ['G', 'G', 'M'] Valgt dør: 2  Premie ved bytte: Mercedes   Premie ved å beholde: Geit
Dørene:  ['G', 'M', 'G'] Valgt dør: 1  Premie ved bytte: Mercedes   Premie ved å beholde: Geit
Dørene:  ['M', 'G', 'G'] Valgt dør: 1  Premie ved bytte: Geit          Premie ved å beholde: Merdedes
Dørene:  ['G', 'M', 'G'] Valgt dør: 2  Premie ved bytte: Geit          Premie ved å beholde: Merdedes
Dørene:  ['M', 'G', 'G'] Valgt dør: 2  Premie ved bytte: Mercedes   Premie ved å beholde: Geit
Dørene:  ['G', 'M', 'G'] Valgt dør: 3  Premie ved bytte: Mercedes   Premie ved å beholde: Geit
Dørene:  ['G', 'G', 'M'] Valgt dør: 1  Premie ved bytte: Mercedes   Premie ved å beholde: Geit
Dørene:  ['G', 'G', 'M'] Valgt dør: 2  Premie ved bytte: Mercedes   Premie ved å beholde: Geit
Dørene:  ['G', 'G', 'M'] Valgt dør: 2  Premie ved bytte: Mercedes   Premie ved å beholde: Geit

Antall Mercedeser ved å bytte:    7 (70.00 %)
Antall Mercedeser ved å stå i ro: 3 (30.00 %)
```