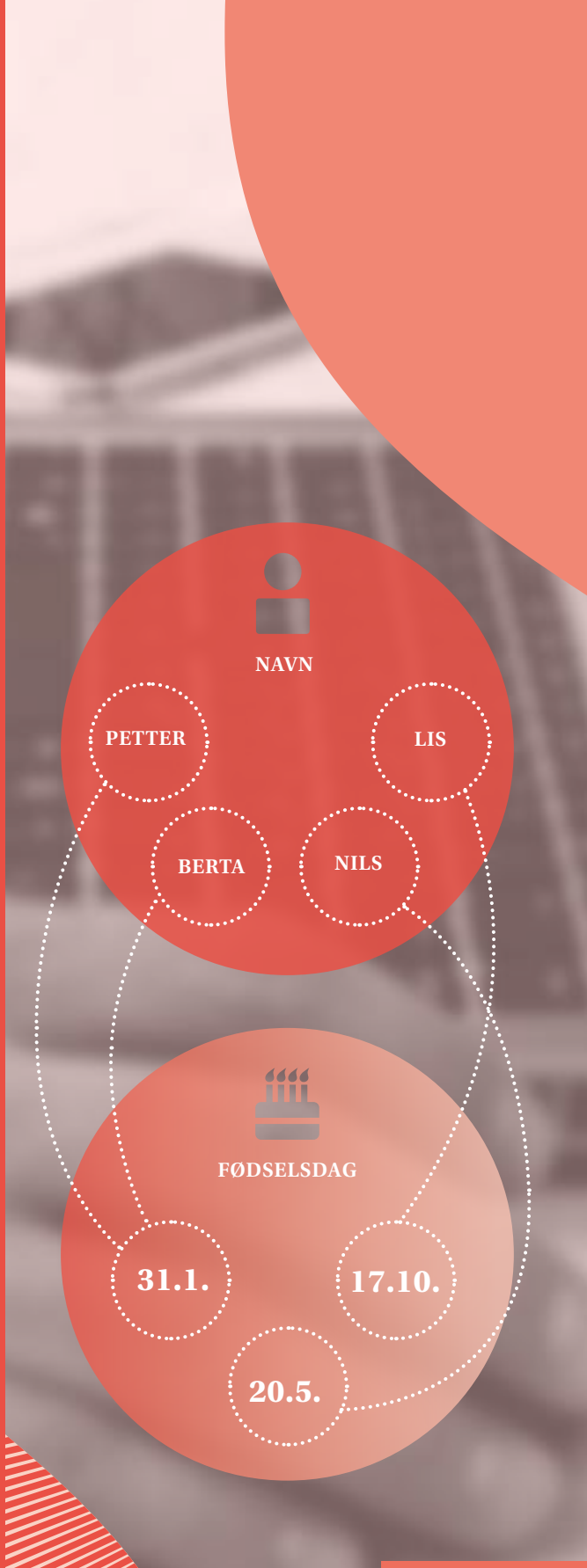
HBV

HØGSKOLEN I BUSKERUD OG VESTFOLD · PROFESJONSHØGSKOLEN ·

Grunnleggende begreper med relevans for informasjonsteknologi

Knut W. Hansson



SKRIFT-SERIEN
Nr. 4

2014

Grunnleggende begreper med relevans for informasjonsteknologi

Knut W. Hansson

Skriftserien fra Høgskolen i Buskerud og Vestfold nr 4 /2014

Om publikasjonen:
Ved Høgskolen i Buskerud og Vestfold, bachelorstudiene i IT, undervises kurset "Grunnleggende programmering" som gir 15 studiepoeng. Seks av disse studiepoengene omhandler grunnleggende begreper med relevans for IT, primært hentet fra matematikk. Publikasjonen inneholder samtlige av forfatterens forelesninger om begreper i det nevnte kurset, samt noe tilleggsstoff for spesielt interesserte.

Om forfatteren:
Knut W. Hansson er førstelektor ved Høgskolen i Buskerud og Vestfold, Campus Ringerike, Handelshøgskolen og fakultet for samfunnsvitenskap.

Grunnleggende begreper med relevans for informasjonsteknologi
Knut W. Hansson

Skriftserien fra Høgskolen i Buskerud og Vestfold nr 4/2014

ISSN: 1894-7522 (online)
ISBN: 978-82-8261-026-1 (online)

Omslag: Maria Prøis Rønneberg

Sammendrag
Ved Høgskolen i Buskerud og Vestfold, bachelorstudiene i IT, undervises kurset "Grunnleggende programmering" som gir 15 studiepoeng. Seks av disse studiepoengene omhandler grunnleggende begreper med relevans for IT, primært hentet fra matematikk. *Denne boken inneholder alle forelesningene mine om begreper i det nevnte kurset, samt noe tilleggsstoff for spesielt interesserte.*

Det nevnte kurset er ett av de første kursene ved studiet og forutsetter bare kunnskaper som inngår i generell studiekompetanse og ingen spesielle kunnskaper hverken i IT eller matematikk.

Kurset "Grunnleggende programmering" skal introdusere studentene til programmering og gi studentene kunnskaper om begreper som de får bruk for gjennom resten av studiet i en rekke andre kurs. Det legges spesiell vekt på å knytte begrepene til anvendelser innen programmering og databaser. F.eks. er mengder og kartesiske produkt, determinander og funksjonell avhengighet grunnleggende for databaseforståelse, mens variable, konstanter og logikk er spesielt viktig i programmering.

Det er laget en egen elektronisk lærerressurs med løsningsforslag til mange av oppgavene.

Forord

Introduksjon til boken

Denne boken inneholder alle mine forelesninger om sentrale begreper i kurset "Grunnleggende programmering" ved Høgskolen i Buskerud og Vestfold. Det skal ikke være nødvendig med ytterligere litteratur for å følge dette kurset. Boken kan også leses på egen hånd.

Uansett om du følger det nevnte kurset eller leser boken på egen hånd, bør du gjøre *alle* oppgavene (det er én til hvert kapittel). Det er ved å *gjøre* det du har lest om, at du lærer best.

Forkunnskaper

Boken forutsetter ingen spesielle kunnskaper innen IT eller matematikk utover det som inngår i generell studiekompetanse med matematikk fra videregående skole. De som har tatt mer enn minimum matematikk fra før, vil nok erfare at boken inneholder mye repetisjon for dem. Allikevel mener jeg boken er nyttig også for dem, fordi den knytter de matematiske begrepene til anvendelse i IT. Det antar jeg vil være nytt for de aller fleste.

Nytteverdi

Informasjonsteknologien – og særlig det teoretiske fundamentet som man finner i faget informatikk – har i stor grad vært utviklet og forsket på av realister. For dem er det naturlig å anvende matematiske ord, definisjoner, bevis, notasjon (skrivemåte) osv. Noen ganger anvendes begreper på en litt annen måte i IT enn i matematikken, men de bygger allikevel på den matematiske forståelsen av ordet. Ordet *informatikk* hevdes faktisk å være avledet av ordene *Informasjon* og *Matematikk*.

I studiene av IT vil dere stadig møte mange slike begreper om og om igjen. Det er derfor effektivt og nyttig en gang for alle å forstå dem ordentlig. F.eks. er *funksjon, variabel og konstant* begreper som dukker opp i både programmering (alle programmeringsspråk), i nettbaserte systemer og i databaser. Forståelse for hva en funksjon, variabel og konstant egentlig er – i matematisk forstand – letter arbeidet med å lære seg de andre emnene. Det er derfor det foreleses om disse begrepene og flere andre helt i starten av studiet.

I denne boken skal jeg se på et utvalg av begreper som jeg synes er spesielt viktige for IT og som mange studenter erfaringsmessig ofte bare har vage forestillinger om. Jeg forsøker å knytte dem til anvendelser innen IT – det er jo det som er hovedfokus i studiet mens de matematiske begrepene egentlig bare er bakgrunnskunnskap.

Innhold

Kapittel 1 – Mengder

Definisjon

På engelsk heter det *Set*. "Studentene i klassen", "Høgskoler i Norge" "Deltakerne i vår gruppe", "Kundene våre" og "Regningene fra i fjor" er alle eksempler på mengder. De kjennetegnes ved

1. De utgjør en *gruppe* som har fått *navn* ("høyskoler i Norge")
2. Alle som hører til gruppen er av samme *type* (høyskoler)
3. Alle i gruppen er *forskjellige* – det er ingen "dubletter", de er "unike" (bare én "Høgskolen i Telemark")
4. Gruppen har ingen *rekkefølge* – de er usortert (det er de samme høyskolene uansett rekkefølge)
5. Det finnes en *forskrift* (en regel) eller en *liste*, som gjør det mulig med sikkerhet å fastslå om et gitt objekt er med i mengden eller ikke ("høyskole er en norsk institusjon som er akkreditert av NOKUT")

De objektene som er med i mengden kalles *elementer*.

Vi kan altså definere en mengde slik:

> **Pugges utenat:**
> ***En mengde er en samling elementer av samme slag uten rekkefølge, der ingen elementer er like. Mengden har navn og det finnes en forskrift/liste som avgjør om et objekt er med i mengden eller ikke.***

Her er noen eksempler:

A={7, 2, 3} – en liste avgrenset med { og }
B=Alle hele tall – uendelig stor mengde
C=alle hester med seks ben – antakelig en tom mengde

Noen tallmengder har fått egne navn, f.eks. N=Naturlige tall {1, 2, 3,…} og N_0 som også har med 0, mens R=alle reelle tall (heltall, desimaltall, brøker, irrasjonelle tall som π o.l.).

Noen begreper for mengder

Kardinalitet

Antallet elementer i en mengde, kalles mengdens kardinalitet, f.eks. er kardinaliteten til tre for mengden {2, 4, 7}.

Mengder uten noen elementer kalles *tomme mengder* og skrives $A = \emptyset$. De har kardinalitet 0.

Likhet

Siden elementene i mengden ikke har noen rekkefølge, blir det meningsløst å spørre hva som er første element, neste/forrige eller siste. To mengder blir like hvis de har de samme elementene:

X={1, 2, 3} og Y={3, 1, 2} er altså like mengder.

Delmengder

Hvis alle elementene i en mengde A også finnes i mengden B, sies A å være en *delmengde* av B. Det skrives slik: $A \in B$. Hvis f.eks. A={11, 12} og B={12, 10, 11} så er $A \in B$.

Dette kan også gjelde for ett enkeltelement. F.eks. er elementet 2 med i {2, 4, 7} og vi kan skrive $2 \in \{2, 4, 7\}$. Elementet 3 er ikke med i {2, 4, 7} og det kan skrives $3 \notin \{2, 4, 7\}$.

Univers

Elementene i en mengde er ofte noen utvalgte elementer fra en større samling. Den store samlingen utgjør da elementenes *domene* som i databaser angir *alle lovlige verdier*. I mengdelæren kalles alle verdier elementene kan ha for elementenes *univers*. Universet kan være uendelig stort, f.eks. alle heltall, resultatet av terningkast o.l., men det kan også være begrenset, f.eks. alle studentene i en klasse.

Hvis ikke hele universet er med i mengden, vil således mengden alltid utgjøre en delmengde av universet. F.eks. vil en gruppe på fem studenter i en klasse på tredve være en delmengde av klassens studenter.

Det er vanlig å betegne universet med bokstaven U, men siden den likner svært på en av operatorene, skriver jeg her heller *univers*.

Tupler

Tupler er også samlinger av flere verdier. I motsetning til mengder, behøver elementene i et tuppel ikke være av samme type og rekkefølgen betyr noe (de utgjør en *sekvens*). De skrives med vanlige parenteser og komma mellom verdiene.

Vi har f.eks. tidligere sett på hvordan punkter angis med x og y-verdi slik (x,y)=(3,6). Det kan altså leses "tuppelet (x,y) er lik tuppelet (3,6)". Siden rekkefølgen betyr noe er (6,3) et annet tuppel enn (3,6).

Andre eksempler på tupler: ("Knut",5), (rød,bjerk). De to verdiene er da åpenbart hentet fra forskjellige mengder.

Mengdeoperasjoner

Enkle operasjoner på én mengde

Når vi har én mengde, kan vi *legge til et element*, *fjerne et element* og *spørre om et element er med i mengden*. Sortering er uaktuelt siden rekkefølgen er uvesentlig[1].

Operasjoner på flere mengder

Når vi har flere mengder, kan vi gjøre operasjoner på dem med operatorer. Disse operatorene utgjør *mengdeoperatorer*. Vi skal se på fem slike:

[1] I databaser har man *mengder* av rader. Rekkefølgen av radene er da uten betydning. Allikevel kan man be om å få radene sortert, men sorteringen gjelder da bare hvordan de vises for en bruker – databasen endres ikke. I databasen sorteres radene aldri.

1. *Union* skrives ∪ og er en binær operator. Vi skriver $A \cup B$.
    a. Union tilsvarer en form for summering, ved at elementene fra B som ikke allerede finnes i A, legges inn i A. Sagt på en annen måte, så legges alle elementene sammen og dubletter fjernes.
    Et eksempel: $\{2,4,7\} \cup \{2,5\} = \{2,4,5,7\}$
    b. Ved union vil antall elementer (kardinaliteten) i svaret bli minst like stort som det er i den med flest.
    $\{2,4,7\} \cup \emptyset = \{2,4,7\}$ eller generelt $A \cup \emptyset = A$ (smlgn med å "legge til 0").
2. *Snitt* skrives ∩ og er også binær. Vi skriver $A \cap B$.
    a. Snitt innebærer å finne de elementene som er med i begge mengdene – de er felles for begge.
    Et eksempel: $\{2,4,7\} \cap \{2,5\} = \{2\}$
    b. Antall elementer (kardinaliteten) i svaret vil bli like stort eller mindre enn i den som har minst.
    $\{2,4,7\} \cap \emptyset = \emptyset$ eller mer generelt: $A \cap \emptyset = \emptyset$
3. *Komplement* skrives ¬ og er unær. Vi skriver $\neg A$. (Dette kan også skrives som en strek over mengdenavnet:$\bar{A}$.)
    a. Komplementet er alle de elementene som *ikke* er med i mengden. Det må følgelig sees i forhold til en større mengde, nemlig alle elementer som finnes av denne typen, dvs. universet.
    b. Anta f.eks. at vi jobber med tallene 1, 2, 3…10 så *univers*={1,2,3…10}. Da er $\neg\{2,4,7\} = \{1,3,5,6,8,9,10\}$.
    c. $\neg\emptyset$ er absolutt alle elementer av denne typen dvs. $\neg\emptyset = univers$.
4. *Differanse* skrives med minustegn. Vi skriver $A - B$.
    a. Differansen mellom to mengder er de elementene som finnes i den ene mengde men ikke i den andre.
    b. F.eks. er $\{2,4,7\} - \{2,5\} = \{4,7\}$
    c. Antallet elementer (kardinaliteten) i svaret blir like stort eller mindre enn antallet i den første.
    d. $\{2,4,7\} - \emptyset = \{2,4,7\}$ eller generelt $\mathrm{A} - \emptyset = \mathrm{A}$ (smlgn med å "trekke fra 0").
5. *Kartesisk produkt* skrives × og er binær. Vi skriver $A \times B$.
    a. Kartesisk produkt er en spesiell måte å gange på. Man lager en *mengde* med *tupler* . Hvert tuppel inneholder ett element fra den første mengden og ett element fra den andre mengden *i denne rekkefølgen*. Vi lager samtlige, mulige kombinasjoner.
    b. F.eks.: $\{2,4,7\} \times \{2,5\} = \{(2,2),(2,5),(4,2),(4,5),(7,2),(7,5)\}$
    c. Merk spesielt at $A \times B \neq B \times A$ fordi rekkefølgen på verdien i tuplene blir motsatt: $\{2,5\} \times \{2,4,7\} = \{(2,2),(2,4),(2,7),(5,2),(5,4),(5,7)\}$
    Det blir like mange tupler men tuplene er forskjellige.
    d. Kardinaliteten i svaret blir kardinaliteten til den ene ganget med kardinaliteten til den andre – i eksemplene 3×2 og 2×3 som begge gir seks tupler.
    e. $\{2,4,7\} \times \emptyset = \emptyset$ og generelt $A \times \emptyset = \emptyset$ (smlgn med å "gange med 0").

Det har ingen mening å spørre om en mengde er *større* enn en annen, man må evt. spørre om *kardinaliteten* er større altså om den ene har flere elementer enn den andre. Det er det vi vanligvis mener i dagligtale også, f.eks. slik at mengden $\{1,2,3\}$ er "større" enn $\{99,100\}$.

Det er også definert hva som skjer hvis de to mengdene har forskjellige typer elementer, f.eks. $\{2, 4, 7\}$ og {rød, grønn} men det tar jeg ikke her, bortsett fra kartesisk produkt. Der er

$$\{2,4,7\}\mathrm{X}\{\text{rød}, \text{grønn}\}$$
$$= \{(2, \text{rød}), (2, \text{grønn}), (4, \text{rød}), (4, \text{grønn}), (7, \text{rød}), (7, \text{grønn})\}$$

Vi kan jo ha forskjellige typer i tupler.

Øvelser i mengdeoperasjoner

Vi trener litt på disse operasjonene ved å løse noen oppgaver.

Union ("eller"):

a) $\{2, 3\} \cup \{5, 7\} = ?$
b) $\{1, 9\} \cup \{7, 3\} = ?$
c) $\{5\} \cup \{9\} = ?$
d) $\{4\} \cup \{4\} = ?$
e) $\{5, 1\} \cup \emptyset = ?$
f) $\{5\} \cup univers = ?$
g) $\emptyset \cup univers = ?$

Huskeregel for union: Logisk $p \vee q$ er sann "hvis den ene eller den andre eller begge er sanne". $A \cup B$ likner og tar med "elementer som finnes i den ene eller den andre eller begge".

Snitt ("og"):

h) $\{3, 5, 9\} \cap \{3, 9\} = ?$
i) $\{1, 2\} \cap \{2, 1\} = ?$
j) $\{9, 5\} \cap \{9\} = ?$
k) $\{1, 5, 2, 7\} \cap \{6, 4\} = ?$
l) $\{3, 2\} \cap \emptyset = ?$
m) $\{2\} \cap univers = ?$
n) $\{rød, grønn\} \cap \{blå, rød\} = ?$

Huskeregel for snitt: Logisk $p \wedge q$ er "sann bare hvis begge er sanne". $A \cap B$ likner og "tar bare med elementer som finnes i begge".

Komplement ("negering"):

Universet er her alle heltall fra 0 til 5, altså $univers = \{0, 1, 2, 3, 4, 5\}$.

o) $\neg\{0, 1, 2, 3\} = ?$
p) $\neg\{5\} = ?$
q) $\neg\emptyset = ?$
r) $\neg univers = ?$

Huskeregel for komplement: Komplementet er "det motsatte". I datamaskiner er universet {0, 1}. Der er komplementet "den motsatte verdien (1 blir til 0 og omvendt). I mengder er komplementet til en mengde lik hele universet *unntatt* de elementene som er med i mengden.

Differanse ("minus"):
Universet er her alle heltall fra 0 til 5, altså $univers = \{0, 1, 2, 3, 4, 5\}$.

s) $\{1, 2, 3\} - \{2, 3\} =?$
t) $\{5, 2, 3, 7\} - \{3, 2\} =?$
u) $\{5, 3\} - \{4\} =?$
v) $\{0, 1\} - \{1, 0\} =?$
w) $\{4,1\} - \emptyset =?$
x) $univers - \{1, 0\} =?$
y) $univers - \emptyset =?$

Huskeregel for differanse: $A - B$ er alle elementene i A *unntatt* de som er med i B. "Trekk fra", altså "ta vekk" elementene som finnes i B.

Kartesisk produkt ("gange"):
z) $\{1, 2\} \times \{3\} =?$
æ) $\{5, 2\} \times \{7, 3\} =?$
ø) Er $\{7,3\} \times \{5,2\} = \{5,2\} \times \{7, 3\}?$
å) $\{0, 1\} \times \{0, 1\} =?$ [kjenner du igjen svaret?]
aa) Er $\{1, 0\} \times \{1, 0\} = \{0, 1\} \times \{0, 1\}?$
bb) $\{2, 3, 4\} \times \emptyset =?$

Huskeregel for kartesisk produkt: Sette sammen to og to – et element fra første mengde og ett fra den andre, på alle mulige måter, så de danner en mengde med tupler. Du skal *ikke* gange!

Kombinasjoner:
Vi antar her at universet er alle heltall fra 0 til 9, altså $univers =$ {0, 1, 2…, 8, 9}.

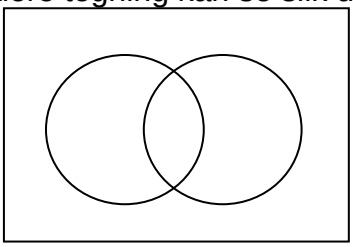
Løs ett uttrykk av gangen, f.eks.: $\{1, 4\} \cup \{2, 4\} \cup \{3\} = \{1, 2, 4\} \cup \{3\} = \{1, 2, 3, 4\}$. Hvis det ikke står parenteser, så løser du fra venstre mot høyre. Hvis det står parenteser, må du løse uttrykket inne i parentesene først.

1) $\{7, 9\} \cup \{6\} \cup \{2, 3\} =?$
2) $\{1, 2\} \cup \{2, 5\} \cup \{1, 7\} =?$
3) $\{0, 3, 2\} \cup \emptyset \cup \{0\} =?$
4) $univers \cup \{3\} \cup \{5\} =?$
5) $(\{8, 7\} \cup \{2, 3\}) \cap \{2, 7\} =?$
6) $\{2, 3\} \cup (\{5, 7\} \cap \{5\}) =?$
7) $univers \cap (\{5\} \cup \{9\}) =?$
8) $(\{0, 1\} \cap \{1\}) \cup \{1, 2\} =?$
9) $\{0, 1\} \cap (\{1\} \cup \{1, 2\}) =?$
10) $\{5, 4\} \cap \emptyset \cup \{5\} =?$
11) $\neg\{0, 1, 2\} \cap \{8, 9\} =?$
12) $\neg((\{0, 1, 2\} \cap \{2,3\}) - \{2, 3\}) \cup \{3, 4, 5\} =?$
13) $\neg(\{0, 1, 4, 8, 9\} \cup \{3, 6, 7, 8, 9\}) \times \{7, 3\} =?$

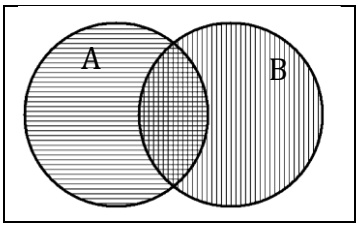
Venn-diagrammer

Ovenfor har vi regnet mye med operatorene, men alle oppgavene gjelder spesielle univers og spesielle mengder. Hvis vi skal vise noe *generelt* nytter det ikke med slike oppgaver. Da må vi være helt generelle. En metode som da er vanlig brukt, er *Venn-diagrammer*[2]. Slike diagrammer bruker et rektangel til å symbolisere et eller annet univers, og sirkler til å symbolisere en eller annen mengde. Hvis det er flere mengder, og de kan ha felles elementer, tegner vi slik:

A og B er to mengder fra universet representert ved rammen rundt, og de to mengdene overlapper. En enklere tegning kan se slik ut:

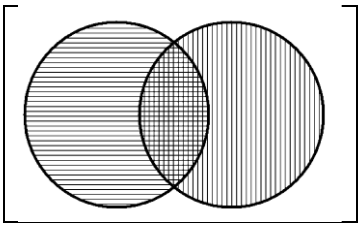


Ved skravering av sirklene, viser vi grafisk hva et uttrykk blir lik. F.eks. kan $A \cup B$ skraveres slik:



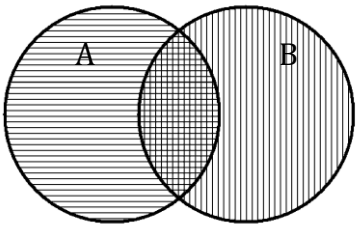
A er skravert vannrett
B er skravert loddrett
$A \cup B$ er *alt som er skravert*

Videre kan $A \cap B$ skraveres slik:



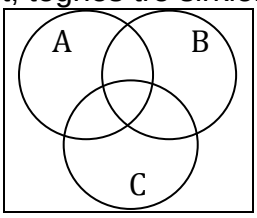
A er skravert vannrett
B er skravert loddrett
$A \cap B$ er *det som er skravert både vannrett og loddrett*

Hvis uttrykket ikke inneholder negering, altså operatoren ¬, eller noe annet som inkluderer universet, er det vanlig å la være å tegne rammen rundt:

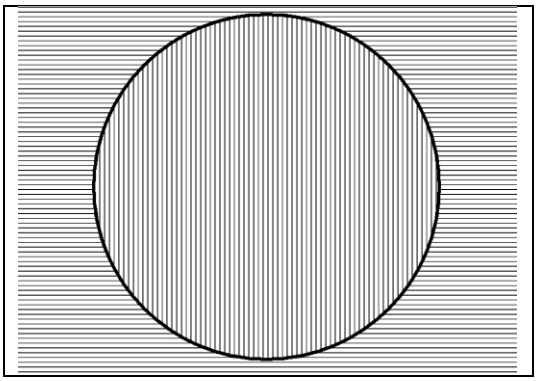


[2] John Venn (1834-1923) var matematiker i Cambridge og arbeidet mye med logikk og sannsynligheter. Han begynte med de diagrammene som nå har navn etter ham.

Hvis tre mengder inngår i uttrykket, tegnes tre sirkler – her med overlapp:



Hvis vi har et uttrykk med ¬, f.eks. ¬*A*, må vi først skravere A og *deretter* kan vi skravere ¬*A*. Da må universet være med:



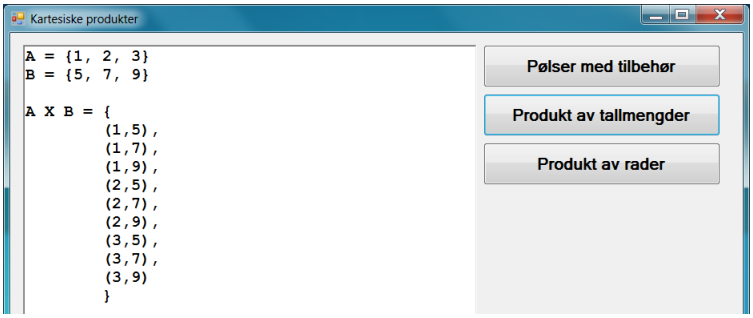
A er skravert loddrett
¬*A* er skravert vannrett

Øvelser i Venn-diagrammer

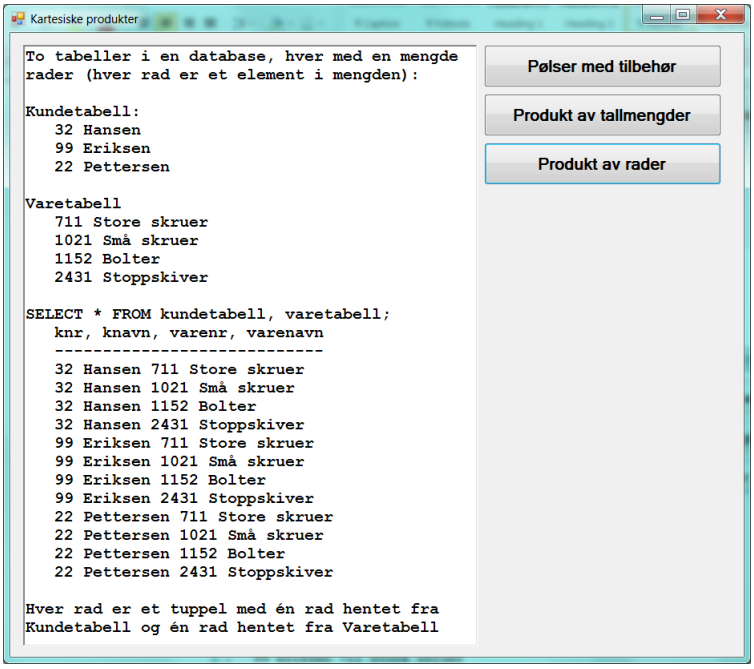
Tegn disse Venn-diagrammene:

1) $A \cup B$ og $B \cup A$. Sammenlikn!
2) Tegn diagrammer og vis at $A \cap \neg B = A - B$.
3) Tegn diagrammer og avgjør om $A \cup \neg \mathrm{B} = \mathrm{univers} - \mathrm{B}$
4) $\neg(\mathrm{A} \cap \mathrm{B})$
5) $\neg(\mathrm{A} \cup \mathrm{B})$
6) $A \cap B \cap C$
7) $(A \cap B) \cup C$
8) $A \cup B \cup \neg \mathrm{C}$
9) $\neg(\mathrm{A} \cap \mathrm{B} \cap \mathrm{C})$
10) Tegn $A \in B$. Hva er da $A \cap B$ og $A \cup B$?
11) Tegn to mengder uten overlapp. Hva er da $A \cap B$?
12) Tegn de to mengdene når $A = B$. Hva er da $A \cap B$ og $A \cup B$?
13) Tegn to diagrammer og sammenlikn $(A \cup B) \cap C$ med $A \cup (B \cap C)$.
14) Gitt $A = \{1, 3\}$ og $B = \{a, c, f\}$. Hva er da $A \times B$?
15) Gitt $A = \{wiener, grill\}$, $B = \{lompe, brød\}$ og $C = \{vann, cola\}$.
    Hva er da $A \times B \times C$?

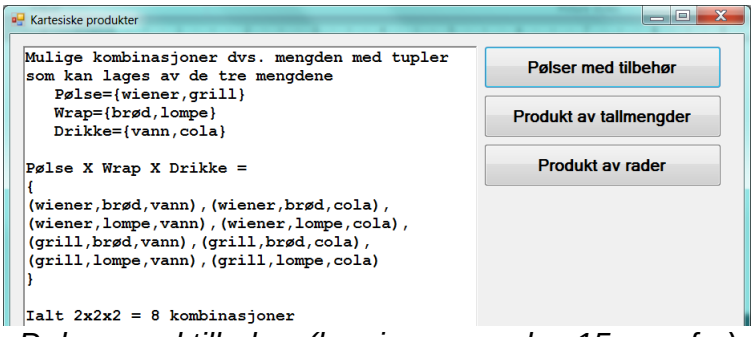
Her tenker jeg å vise programmet "Kartesisk produkt" som beregner kartesiske produkter, inkludert "pølseoppgaven" ovenfor. Programmet produserer følgende skjemaer:



Kartesisk produkt av to tallmengder



Resultatet av en SQL-spørring mot en database. Resultatet er et kartesisk produkt.



Pølser med tilbehør (løsning av øvelse 15 ovenfor)

Oppgave til kapittel 1 (mengder)

A, B og C er mengder. Her skriver jeg komplementet ("ikke") med tegnet ¬, f.eks. ¬A.

Oppgave 1

Gitt at

1) Mengden G = ”Guttene i klassen”
2) Mengden J = ”Jentene i klassen”
3) G ∩ J = Ø

Tegn Venndiagram for denne klassen. Hva betyr punkt 3 på vanlig norsk?

Oppgave 2

Vis med to Venndiagrammer at

¬(A ∩ B) = ¬A U ¬B

(Dette er en av de Morgans lover)

Oppgave 3

Vis med to Venndiagrammer at

¬(A U B) = ¬ A ∩ ¬ B

(Dette er en annen av de Morgans lover)

Oppgave 4

Gitt to mengder ”Ryk” og ”Reis”.

Vis ved Venndiagram at ”ikke Ryk og ikke Reis” er noe helt annet enn ”ikke (Ryk og Reis”).

Oppgave 5

Tegn et Venndiagram med de tre mengdene ”Kjærlighet”, ”Ekteskap” og ”Samhold”. Vær kreativ og sett et passende navn på alle snittene (f.eks. ”Kjærlighet ∩ Ekteskap”), de som er snitt mellom to mengder og den som er snitt mellom alle tre mengdene.

Oppgave 6

Vis med to Venndiagrammer at

A ∩ (B U C) = (A ∩ B) U (A ∩ C)

(Distributiv lov)

Oppgave 7

Vis at

A U (B ∩ C) = (A U B) ∩ (A U C)

(Distributiv lov)

Oppgave 8
Gitt tre mengder:
A = ”Menn”
B = ”Smarte”
C = ”Blonde”
Tegn og studer ett Venndiagram. Forklar så med ord (uten å tegne flere Venndiagrammer) hvem som hører med i kategoriene

1. ¬A
2. A ∩ C
3. B U C
4. ¬(A ∩ B)
5. ¬A U ¬B
6. ¬A ∩ ¬B ∩ C

Sammenlign 4 og 5.

Oppgave 9
En delmengde innebærer at alle elementene i den ene mengden er med i den andre. Gitt at en mengde A er en delmengde av B. Tegn Venndiagram. Finn et eksempel på hva mengdene A og B kan være.

Oppgave 10
Gitt mengdene
A = {3, 8, 7, 9, 5}
B = {8, 5, 9, 2}
der A og B er delmengder av S = {1, 2, 3, …,10},
Finn de tre mengdene (A U B), (A ∩ B) og (¬A ∩ B).

Oppgave 11
Gitt at A={Stor, Middels, Liten} og B={Hund, Katt}, hva er da det kartesiske produktet A x B?

Oppgave 12
Hvis du vil ha mer, så se på http://fag.grm.hia.no/fagstoff/perhh/htm/fag/stat/ff0140/o/k01/k01.htm og løs de tre første oppgavene (01_01 til 01_03). Løsningen finner du på http://fag.grm.hia.no/fagstoff/perhh/htm/fag/stat/ff0140/l/k01/k01.htm.
(Generelt er http://fag.grm.hia.no/fagstoff/perhh/htm/fag/stat/ff0140/K01.htm en fin ressurs.)

Kapittel 2 - Logikk

Utsagn

Logikken baserer seg på *utsagn* (statements) som enten er *sanne* eller *usanne* (true/false). Det skal kunne avgjøres objektivt og uten tvil om utsagnet er sant eller ikke. Noen eksempler:

a. Har du det bra?
   Dette er *ikke* et utsagn som er sant/usant men et spørsmål.
b. Hun er pen!
   Dette er upresist, for hvem er "hun"? Og om det er sant eller usant baserer seg på en personlig vurdering (det er subjektivt). Det er derfor *ikke* et utsagn.
c. Det finnes liv på andre planeter!
   Dette er det – foreløpig - umulig å avgjøre. Det er derfor *ikke* et utsagn.
d. -10 er mindre enn -7!
   Dette kan avgjøres objektivt og er et utsagn (det er forresten sant).
e. X > 7 (der X er et tall)
   Dette kan bestemmes objektivt avhengig av verdien av X og er et utsagn

Utsagn representeres vanligvis med *p*, *q*, *r* osv. Her er noen utsagn:

p: Læreren L er førstelektor
q: Y er en planet i vårt solsystem
r: Tallet x > 7

Da kan vi skrive

Både *p* er sann og *q* er sann skrives slik: $p \wedge q$ *(konjunksjon)*
Enten *q* er sann eller *r* er sann eller begge to er sanne skrives slik: $q \vee r$ (*disjunksjon*)
Det motsatte av *r* skrives slik: $\neg r$ (eller slik: $\bar{r}$) (*negasjon*)

Videre brukes parenteser som i matematikken ellers: $(p \vee q) \wedge r$ tilsvarende $(2 + 3) \cdot 6$ i matematikken.

Sannhetsverditabeller (truth tables)

Når vi har to utsagn som vi ikke vet sannhetsverdien for, kan vi sette opp en tabell som viser alle muligheter. Da skrives gjerne 0 for usann og 1 for sann[3].

p	q	$\neg p$	$p \wedge q$	$p \vee q$
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

I eksemplet har vi to utsagn *p* og *q*. Vi har satt opp de fire mulighetene rad for rad (p er usann og q er usann, p er usann men q er sann osv.). For hver kombinasjon har vi

[3] Liknende er det også i Visual Basic, der 0=True og alt annet er False. Settes noe til False får det verdien -1. Teknisk er 0 lik 0000 0000…0000. Det motsatte er -1 som er lik 1111 1111…1111. Altså er True og False komplementer av hverandre. Det å gjøre om et maskinord til komplementet av det samme ordet, er *meget* raskt gjort i maskinen – det gjøres direkte i elektroniske kretser. Uttrykket *Not True* utføres altså ved å finne komplementet (det motsatt = Not) av True.

laget en kolonne for $\neg p$ og kombinasjoner av *p* og *q*. Vi kan da se f.eks. av merkede raden at hvis det er slik at *p* er sann og *q* usann, så er $\neg p$ usann, $p \wedge q$ usann mens $p \vee q$ er sann. Kolonne tre til fem er dessuten definisjonen av hva vi mener med de tre operatorene.

Legg merke til at de to første kolonnene til sammen utgjør binære tall fra 00 til 11. Det blir fire muligheter fordi det er to utsagn som begge kan være sann og usann. Hvis det er tre utsagn, blir det da 2^3=8 muligheter osv. Tabellen blir fort store!

Dine oppgaver vil gå ut på å sette opp sannhetsverditabeller slik at vi får avgjort når et uttrykk er sant og usant. Du må da begynne med de enkle utsagnene og så sette dem sammen fra "innerst" (innerste parentes) til "ytterst" inntil du får en kolonne med det søkte uttrykket. Du må bare bruke én operator av gangen!

Eksempel:
Sett opp sannhetsverditabell for $(p \wedge q) \vee \neg p$.

Løsning:

p	q	$\neg p$	$p \wedge q$	$(p \wedge q) \vee \neg p$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	0
1	1	0	1	1

Øvelser

a. $p \wedge \neg q$
b. $p \vee \neg q$
c. $\neg p \wedge \neg q$
d. $\neg(p \vee q)$
e. $\neg p \vee \neg q$
f. $\neg(p \wedge q)$
g. $\neg p \vee q$
h. $p \wedge (q \vee r)$

Av sannhetsverditabellene kan vi se at hvis vi hadde skrevet dem inn i samme diagram, ville oppg c = oppg g, og e = f. Av dette kan vi slutte at følgende alltid gjelder:

1. $\neg(p \wedge q) = \neg p \vee \neg q$
2. $\neg(p \vee q) = \neg p \wedge \neg q$

Dette er to av Augusta de Morgans[4] lover. Vi skal ikke se mye på slike lover her.

Operander og operatorer

I et uttrykk som

$$\{x[2 \cdot (3 + 5) - 6] : 2\}$$

er tallene *operander* og tegnene *operatorer*. Her er det *aritmetriske operatorer*.

[4] A. de Morgan jobbet i London på 1800-tallet som matematiker. Han var venn og lærer for Charles Babbage, lady Ada Lovelace og andre.

I et uttrykk som

$$p \wedge (q \vee \neg r)$$

er bokstavene operander og de andre tegnene er *logiske operatorer*. Slike operatorer kalles også Boolske (Boolean) operatorer etter George Boole som skrev bøker om logikk rundt 1850.

I uttrykk som

$$x > 5$$
$$alder \leq 50$$
$$inntekt = 103\,000$$

er det brukt *sammenlikningsoperatorer* (relasjonelle operatorer).

Spesielle maskinoperatorer

I datamaskiner benyttes også

NAND =$\neg(p \wedge q)$

NOR = $\neg(p \vee q)$

XOR = $(p \wedge \neg q) \vee (\neg p \wedge q)$ som innebærer at bare én av *p* og *q* er sann.

Videre finnes AND, OR og NOT som kan kjøpes ferdig og brukes til å bygge datamaskiner av, fra bunnen. Lykke til☺! I datamaskiner konstrueres logiske operatorer med hardware og blir meget raske.

Poenget med alle disse maskinoperatorene er at de kan brukes i beregninger, f.eks. er

$p\ XOR\ q$ er det samme som *p*+*q* når *p* og *q* er enkle bits, etter følgende tabell:

p	*q*	*p+q*	*p XOR q*
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0 (med overflow)	0

Med en XOR-krets kan altså maskinen summere enkle bits. Det finnes flere andre typer operatorer og dere vil møte på dem etter hvert.

Logiske valg i Visual Basic

Logiske valg i VB finnes i flere typer setninger, både de som velger mellom flere handlinger (*if then*) og de som skal gjenta noe så lenge noe er sant (*while*). VB bruker de logiske operatorene NOT, AND, OR, XOR og parenteser. Verdien heter *True* og *False*. Man bruker også sammenlikningsoperatorer, f.eks. > og <=.

Noen eksempler:

```
If navn = "knut" Then
   MsgBox("Fint navn!")
End If

If pris >= 1000 Then
   MsgBox("Husk å gi rabatt!")
End If
```

Hva er forskjellen på disse to:

```
If navn <> "Knut" Then
   MsgBox("Dummen!")
End If
If Not (navn = "Knut") Then
   MsgBox("Dummen!")
End If
```

Øvelse

Hva er forskjellen på disse to (*Enabled* er en Boolsk verdi):

```
If cmdAvslutt.Enabled = True Then
   'gjør noe
End If
If cmdAvslutt.Enabled Then
   'gjør noe
End If
```

Generelt unngår vi å skrive =*True* og =*False* i logiske uttrykk. Det er unødvendig.

Hva gjør denne setningen:

```
cmdAvslutt.Enabled = Not cmdAvslutt.Enabled
```

Ekstra 1: "Kortslutning" av logiske uttrykk i VB

AndAlso

Fra logikken vet vi at i et uttrykk som i denne if-setningen

```
If alder > 67 And kjønn = kvinne Then
```

så vet vi at hvis alder *ikke* er større enn 67, så er utrykket usant, uansett hva kjønn måtte være. I så fall har det ingen hensikt å sjekke om kjønn = kvinne for uttrykket blir ikke sannere av det. Det er mulig å få VB til å stoppe så snart den har oppdaget at alder ikke er større enn 67 ved å skrive *AndAlso*:

```
If alder > 67 AndAlso kjønn = kvinne Then
```

Dette kalles å "kortslutte" det logiske uttrykket ("short circuiting"). Det er spesielt nyttig hvis det er viktig å ikke kontrollere *kjønn* hvis ikke *alder* er stor nok fordi brukeren bare oppgir kjønn for eldre personer (*kjønn* har altså bare fått verdi for eldre mennesker i dette programmet).

OrElse

Tilsvarende for OR:

```
If alder > 67 Or kjønn = kvinne Then
```

Dette uttrykket vet vi er sant hvis *alder* > 67 uansett hva *kjønn* er. Vi kan "kortslutte" også dette uttykket med *OrElse*:

```
If alder > 67 OrElse kjønn = kvinne Then
```

Det vil være nyttig hvis *kjønn* ikke har fått verdi for eldre mennesker, det er bare for yngre at kjønn er oppgitt av brukeren.

Xor

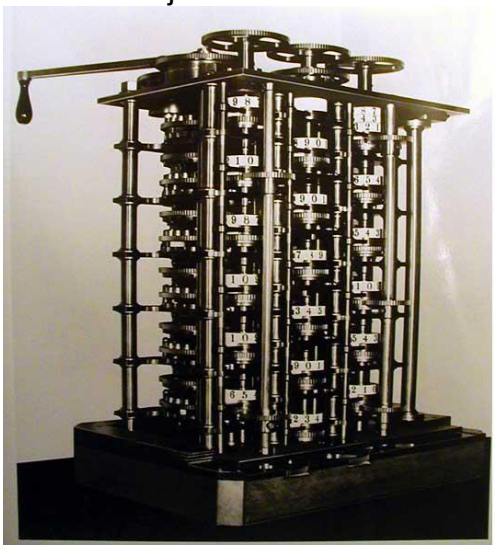
Visual Basic (og mange andre språk) har også den logiske operatoren Xor. Denne innebærer at bare én av de to skal være sanne:

```
If alder > 67 Xor kjønn = kvinne Then
```

Dette utrykket blir sant hvis alderen er høy nok eller kjønn er kvinne, men ikke hvis begge inntreffer samtidig (eldre får vanligvis spesialbehandling og kvinner får det vanligvis også, men eldre kvinner får det ikke).

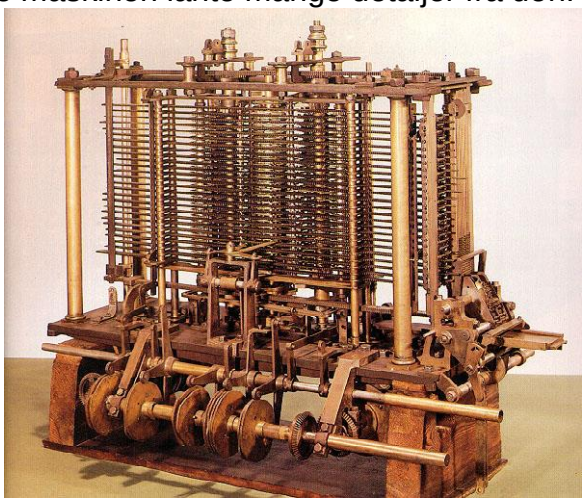
Ekstra 2: Kort om Charles Babbage

Charles Babbage, matematiker fra Cambridge, er i dag mest kjent for sin konstruksjon – en mekanisk computer. Han laget først en kalt "the difference engine" som beregnet tabeller for visse funksjoner.



The difference engine

Senere konstruerte han (men klarte ikke å lage pga for dårlig teknologi, unøyktigheter i produksjonen av tannhjul osv.) ca 1834 "the analytical engine". Den er i struktur svært lik dagens datamaskiner. Maskinen er delvis laget i våre dager og den første elektroniske maskinen lånte mange detaljer fra den.



The analytical engine

Beskrivelsen ble publisert på fransk og senere oversatt av Ada Lovelace. Hun la selv til ting, som f.eks. programmet for beregning av Bernouilli-tallene – en nokså komplisert rekke. Dette regnes av mange som "verdens første program".

Ekstra 3: Kort om Lady Ada Lovelace

Ada (1815-1852) ble kjent som "den første programmerer" og har fått programmeringsspråket ADA oppkalt etter seg. Hun var datter av dikteren Lord Byron og gift med Lord Lovelace.

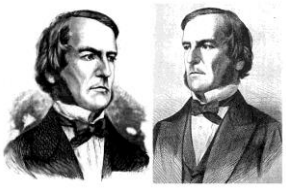
Ada var venn av Charles Babbage og studerte bl.a. under/med George Boole (samme alder som henne og kjent for Boolsk algebra og ”difference equations”).

En referanse (http://www.ex.ac.uk/BABBAGE/ada.html) hevder riktignok at

> *It is often suggested that Ada was the world's first programmer. This is nonsense: Babbage was, if programmer is the right term. After Babbage came a mathematical assistant of his, Babbage's eldest son, Herschel, and possibly Babbage's two younger sons. Ada was probably the fourth, fifth or sixth person to write the programmes. Moreover all she did was rework some calculations Babbage had carried out years earlier. Ada's calculations were student exercises. Ada Lovelace figures in the history of the Calculating Engines as Babbage's interpretress, his `fairy lady'. As such her achievement was remarkable.*

Mitt syn er at myten om at en *kvinne* var verdens første programmerer er alt for spennende til å ødelegges av slike trivialiteter☺!

Oppgave til kapittel 2 (logikk)



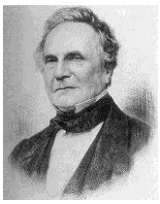
Boole

de Morgan (studentkarikatur)

Oppgave A

Lag sannhetsverditabell for de nedenstående logiske uttrykk.

1. $p \wedge q$ og $q \wedge p$. Sammenlikn de to (den kommutative lov)!
2. $p \vee q$ og $q \vee p$. Sammenlikn de to (den kommutative lov)!
3. $(p \wedge q) \wedge r$ og $p \wedge (q \wedge r)$. Sammenlikn de to (den assosiative lov)
4. $(p \vee q) \vee r$ og $p \vee (q \vee r)$. Sammenlikn de to (den assosiative lov)
5. $p \wedge True$ og p. (Det første uttrykket innebærer at p konjugeres med noe som "alltid er sant": "p OG alltid sant".) Sammenlikn og forklar!
6. $p \vee False$ og p. (Det første uttrykket innebærer at p disjunkeres med noe som "alltid er usant": "p ELLER usant".) Sammenlikn og forklar!
7. $\neg\neg p$ og p (p er negert to ganger = "ikke-ikke-p"). Sammenlikn og forklar!
8. $p \wedge p$ og p. Sammenlikn og forklar!
9. $p \vee p$ og p. Sammenlikn og forklar!
10. $p \vee \neg p$. Hva finner du? Forklar!
11. $p \wedge \neg p$. Hva finner du? Forklar!
12. $p \vee True$. Hva finner du? Forklar!
13. $p \wedge True$. Hva finner du? Forklar!



Babbage

Lady Lovelace

Oppgave B

Lag sannhetsverditabell for uttrykkene under:

1. $(p \wedge q) \vee r$
2. $p \wedge (q \vee r)$
3. Sammenlikn resultatene i spørsmål (1) og (2). Ble de like? Kan vi flytte parentesen dit vi vil? Hva er forskjellen på disse to sammenliknet med den assosiative loven i oppgave A?
4. $(p \vee q) \wedge (p \vee r)$
5. Studer sannhetsverditabellen til spørsmål (4). Hva skal egentlig til for at den blir sann? Prøv å skrive det enklere (altså lage et enklere uttrykk med samme sannhetsverditabell).

Kapittel 3 – Tallsystemer

Potenser

For å forstå tallsystemer, må man kjenne til potenser. I desimalsystemet (titallssystemet") som vi bruker daglig har vi f.eks.

$$10^4 = 10 \cdot 10 \cdot 10 \cdot 10 = 10\,000\ (med\ fire\ nuller)$$

som leses "10 opphøyd i fjerde potens" eller kortere "ti opphøyd i fjerde" eller enda kortere "ti i fjerde". Dvs. 10 ganget med seg selv fire ganger.

$$10^3 = 10 \cdot 10 \cdot 10 = 1\,000\ (med\ tre\ nuller)$$
$$10^2 = 10 \cdot 10 = 100\ (med\ to\ nuller)$$
$$10^1 = 10 = 10\ (med\ én\ null)$$
$$10^0 = 1 = 1\ (med\ ingen\ nuller)$$

Generelt er

$$a^b = \underbrace{a \cdot a \cdot a \ldots}_{a \text{ gjentatt } b \text{ ganger}}$$

altså *a* ganget med seg selv *b* ganger. *a* kalles da grunntallet og *b* kalles eksponenten.

Spesielt bør du merke deg at $a^1 = a$ for enhver *a* og $a^0 = 1$ for enhver *a*.

Hvis grunntallet *a* = 2 får vi følgende potenser

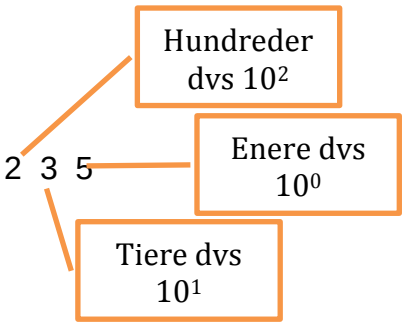
$$2^0 = 1$$
$$2^1 = 2$$
$$2^2 = 4$$
$$2^3 = 8$$
$$2^4 = 16$$
$$\ldots$$

Disse kalles "toerpotenser".

Tallsystemer

Desimalsystemet

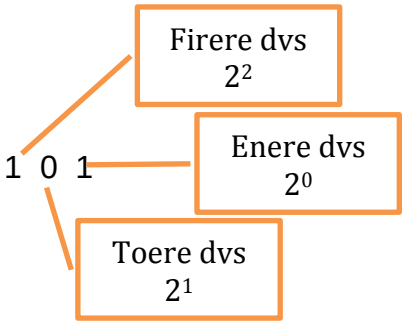
Av *deka* – gresk for ti
= Titallssystemet
Grunntall 10 dvs. "vi teller ti og ti".
10 forskjellige sifre: 0, 1, 2, 3…9
Posisjonelt tallsystem.



Tallet leses da også "to hundrede og tretti fem". Sifrene blir mer verd jo lengre mot venstre i tallet de står, faktisk er et siffer alltid 10 ganger så mye verd som sifferet til høyre.

Binært tallsystem

Av *bi-* gresk for todelt
= Totallssystemet
Grunntall 2 dvs. "vi teller to og to".
2 forskjellige sifre: 0, 1
Posisjonelt tallsystem.



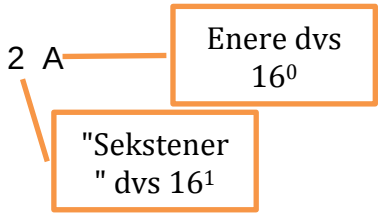
Tallet kan ikke leses på vanlig måte da språket vårt ikke har passende ord for det. Vi leser isteden "binært en null en". Sifrene blir mer verd jo lengre mot venstre i tallet de står, faktisk er et siffer alltid 2 ganger så mye verd som sifferet til høyre.

Ett binært siffer kalles en *bit* (av *binary digit* = binært siffer)

Jeg har en "dings" som viser telling binært. Den tenker jeg å vise.

Heksadesimalt tallsystem

Av *hexadeca* gresk for 16
= Sekstentallssystemet
Grunntall 16 dvs. "vi teller seksten og seksten".
16 forskjellige sifre: 0, 1, 2, 3…9, A, B, C, D, E, F
Posisjonelt tallsystem.



Tallet kan ikke leses på vanlig måte da språket vårt ikke har passende ord for det. Vi leser isteden "hexa to A". Sifrene blir mer verd jo lengre mot venstre i tallet de står, faktisk er et siffer alltid 16 ganger så mye verd som sifferet til høyre.

Andre, posisjonelle tallsystemer

På samme måte kan vi lage tallsystemer med ethvert, positivt heltall større enn én (hvorfor positivt? hvorfor større enn én?).

Øvelse
Beskriv et tallsystem basert på grunntallet 8 på samme måte som ovenfor. Det har vært i vanlig bruk i IT og kalles det oktale (av octo = 8) tallsystemet.

Romertall

Romertallene er ikke posisjonelle. Isteden avgjøres det om noe skal legges til eller trekkes fra, avhengig av sifrene som står rundt:

Eksempel:
LIIX (L betyr 50, I betyr 1 og X betyr ti)

Dette tallet skal forståes slik: Begynne med femti. Det som står bak er mindre og da skal det legges til. Hvor mye skal legges til? Jo ti med fradrag av to enere fordi de står foran et større siffer ti. Altså får vi (50 + (10 – 1 – 1)) = 50 + 8 = 58. Disse tallene er greie å forstå, men håpløse å regne med. Romerne var da heller ikke kjent for å utvikle matematikken noe særlig. I informatikk brukes aldri slike tall.

Presisering

Det kan være usikkert hvilket tallsystem vi bruker. Da kan vi legge til en indeks bak som angir tallsystemet, f.eks.

235_{10} eller 235_D
101_2 eller 101_B
$2A_{16}$ eller $2A_H$

Omregning mellom noen tallsystemer

Et tall er en verdi, et antall av noe. Det endrer ikke verdi om det skrives desimalt, binært, heksadesimalt eller med noe annet grunntall, evt. med romertall eller med bokstaver. Det er bare skrivemåten som endres. Noen ganger er det nyttig å se tallet binært, andre ganger er heksadesimalt bedre. Jeg viser derfor nedenfor noen høyst aktuelle omregninger mellom tallsystemer.

Da man brukte åttebits maskiner var det oktale tallsystemet mye brukt. Nå er det det heksadesimale som brukes mest, i tillegg til det binære som best tilsvarer maskinens virkemåte.

Heksadesimalt til binært

Hvis vi ser litt på en "oversettelse" fra binære til heksadesimale tall, kan vi sette opp denne tabellen:

$1_H = 0001_2$
$2_H = 0010_2$
$3_H = 0011_2$
$4_H = 0100_2$
...
$D_H = 1101_2$
$E_H = 1110_2$
$F_H = 1111_2$

Altså: Hvert heksadesimale siffer blir akkurat fire binære siffer.

Noen eksempler på omgjøring:
3 5 $_H$ = 0011 0101$_B$
2 7 $_H$ = 0010 0111$_B$
6 D $_H$ = 0110 1101$_B$
A F 2 4$_H$ = 1010 1111 0010 0100$_B$ (fire bytes = 16 bits)
Hvis vi viser et binært tall som heksadesimalt isteden, reduserer vi sterkt antall siffer. Hvis du tenker deg at det siste eksempelet er en feilmelding fra systemet som du skal slå opp i en manual, så synes nok du også at det er enklere å få den heksadesimalt enn binært!

Øvelser
Skriv dette binært med 8 binære sifre (en byte):
7 8$_H$
6 2$_H$
1 5$_H$
8$_H$
F F$_H$

Jeg tenker å vise et program jeg har laget som teller binært og omgjør til desimalt og heksadesimalt.

Binært til heksadesimalt
Hvis vi skal omgjøre fra binært til heksadesimalt tall, grupperer vi de binære sifrene fire og fire bakfra, og hver gruppe blir ett heksadesimalt siffer, f.eks.
1101 1110$_B$ = DE$_H$
101 1010$_B$ = 5A$_H$ (legg til en tenkt null foran – en null foran endrer ikke verdien)

Øvelser
Gjør om disse binære tallen til heksadesimale:
11011$_B$
11111111$_B$

Binært til desimalt
Dette blir litt mer komplisert, men fortsatt ganske greit.

For det første kan dette gjøres enkelt med kalkulatoren i Windows. Det tenker jeg å vise.

For det andre kan det, med litt forståelse, gjøres manuelt med enkel regning:
$1011_B\ betyr\ 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 8 + 2 + 1 = 11_D$
11001$_B$ = 16+8+1=25$_D$
11111111$_B$ = 128+64+32+16+8+4+2+1=255$_D$
eller enklere:
11111111$_B$ = 100000000$_B$-1 = 256-1 = 255$_D$

Øvelser
Skriv disse binære tallene desimalt:
11011$_B$
11100 $_B$
10001 $_B$
1101101 $_B$

Hva er det største desimale tall vi kan skrive med 8 bits?

Desimalt til binært

Her benytter vi heltallsdivisjon slik som i dette eksempelet:

Hva er 17_D binært?

Beregn		*Svar*	*Rest*
17_D : 2 =	8	1	
8 : 2 =	4	0	
4 : 2 =	2	0	
2 : 2 =	1	0	
1 : 2 =	0	1	
0 : 2 =	0	0	
0 : 2 =	0	0	

… osv

Svaret leses oppover = …0010001_B = 10001_B

Øvelser

Skriv nedenstående desimale tall binært:

32_D

71_D

127_D

Negative tall i datamaskinen

Anta – for enkelhets skyld – at vi har en firebits maskin. Det er altså en "nibble"-maskin (en vits da "nibble" betyr en liten smakebit, altså på engelsk en "small byte").

Positive tall skrives som de er, f.eks. 7_D = 0111 i maskinen. Negative tall er det verre med. Vi kan ikke ha -0111 i maskinen for minustegn finnes jo ikke. Isteden kan vi prøve å sette av første (mest signifikante) bit til fortegn: 0=+ og 1=minus. Da får vi -7_D=1111 i maskinen. Det kan se ut til å fungere helt fint, men reglene for summering blir plutselig feil:

-7+7=0_D blir til 1111+0111=0110 i maskinen (pluss en ekstra bit som blir "overflow"), og svaret på -7+7 blir ikke null! Det blir ikke riktig.

Isteden bruker man en teknikk kalt "toerkomplement" slik:

+7	=	0111
Komplement:	1000	
Legg til 1:	1	
Resultat		1001

Med dette systemet, vil altså -7 representeres med toerkomplement som 1001. La os da prøve regneregelen og legge sammen -7 og +7:

-7	1001
+7	0111
Sum	1)0000 (samt en "overflowbit" som maskinen ser bort fra)

Svaret på -7+7 ble nå null og regnereglene stemmer.

Øvelse
Negative tall
(a) En firebits maskin bruker toerkomplement for negative tall.
Vis hvordan vi beregner -2+7?
3-5?
-2-4?

(b) Hvis vi har en åttebits maskin utvider vi bare bitmønsteret.
Hva blir da -1?
-128?
-3?

Logikk på bitnivå

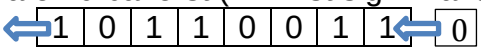
"Bitmønstre"

Et maskinord på fire bytes (32 bits) eller åtte bytes (64 bits) kan representere et tall, et tegn, en farge, en maskininstruksjon og mye annet. Vi kan også velge å se på det som et *bitmønster*. Da ser vi på én og én bit og sier at den er "satt på" hvis den er 1 og er "skrudd av" hvis den er 0.

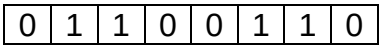
For enkelhets skyld ser vi her på maskinord som bare er én byte, men diskusjonen blir nøyaktig lik for større maskinord.

Shift-operasjoner

Vi kan flytte alle bits én plass mot venstre. Den fremste biten ("mest signifikante bit") kastes ut og det fylles på må en 0 bakerst (i "minst signifikante bit"):

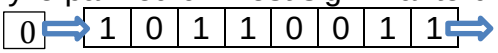


Resultatet blir et nytt bitmønster:

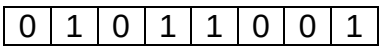


Det kalles et "venstreskift" ("left shift").

Tilsvarende kan vi skifte mot høyre. Da forkaster vi minst signifikante bit, flytter alle de andre mot høyre og fylle på med 0 i mest signifikante bit:



Resultatet blir et nytt bitmønster:

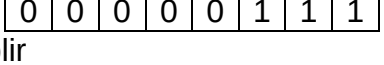


Dette er et "høyreskift".

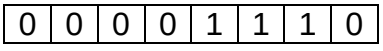
I Visual Basic skrives skiftoperasjoner slik:

```
Dim mønster As Byte = 7
mønster = mønster << 1
```

mønster er altså en byte med verdien 7. I andre linje skifter vi mønster ett hakk mot venstre. Resultatet blir at mønster blir 14. Legg merke til at det blir akkurat dobbelt så stort – hvorfor? Det opprinnelig bitmønsteret var 7 binært, altså



Resultatet etter venstreskift blir



som er 14 (8+4+2).

Tilsvarende kan vi høyreskifte med operatoren >> slik

```
Dim mønster As Byte = 7
mønster = mønster >> 1
```

Hva blir resultatet da? Tegn opp selv.

Vi kan skifte mer enn én plass også:

```
Dim mønster As Byte = 7
mønster = mønster >> 3
```

Da flyttes alle bits tre plasser mot høyre. Resultatet blir 0. Tegn opp selv og kontroller.

Logiske operatorer

Vi kan bruke logiske operatorer (*And*, *Or*, *Xor*, *Not*) også på bitnivå. La oss begynne med *Not*.

Not

Å gjennomføre nektelsen *Not* på bitnivå skaper komplementet – alle 0 blir til 1 og omvendt:

a (=89):	0	1	0	1	1	0	0	1
Not a:	1	0	1	0	0	1	1	0

Først bit er 0 – blir 1.
Andre bit er 1 blir 0.
osv. bit for bit.

I VB skriver vi

```
Dim resultat As Byte
Dim a As Byte = 89
resultat = Not a
```

Resultatet blir 166 (128+32+4+2).

And

Hvis jeg bruker *And* mellom to bitmønstre, skal de to bitmønstrene sammenliknes posisjon for posisjon. Hvis begge bitene er 1 så blir resultatet 1, eller blir det 0. Dette er nøyaktig det samme som i en sannhetsverditabell.

a (=89):	0	1	0	1	1	0	0	1
b (=55):	0	0	1	1	0	1	1	1
a And b:	0	0	0	1	0	0	0	1

Første bit er 0 i begge – resultatet er 0.
Neste bit er 1 i a og 0 i b – resultatet er 0.
Tredje bit er 0 i a og 1 i b – resultatet er 0.
Fjerde bit er 1 i begge – resultatet er 1.
osv. bit for bit.

I VB:

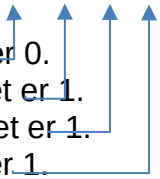
```
Dim resultat As Byte
Dim a As Byte = 89
Dim b As Byte = 55
resultat = a And b
```

Resultatet blir 17 (16+1).

Or

Når jeg anvender *Or* mellom to bitmønstre, skal de to bitmønstrene sammenliknes posisjon for posisjon. Hvis minst én av bitene er 1 skal resultatet bli 1 (som i logikken):

a (=89):	0	1	0	1	1	0	0	1
b (=55):	0	0	1	1	0	1	1	1
a Or b:	0	1	1	1	1	1	1	1



Første bit er 0 i begge – resultatet er 0.
Neste bit er 1 i a og 0 i b – resultatet er 1.
Tredje bit er 0 i a og 1 i b – resultatet er 1.
Fjerde bit er 1 i begge – resultatet er 1.
osv. bit for bit.

I VB:

```
Dim resultat As Byte
Dim a As Byte = 89
Dim b As Byte = 55
resultat = a Or b
```

Resultatet blir 127 (64+32+16+8+4+2+1).

Xor

Med *Xor* skal resultatet bli 1 bare hvis akkurat én av de to bitene er 1.

a (=89):	0	1	0	1	1	0	0	1
b (=55):	0	0	1	1	0	1	1	1
a Or b:								

I VB:

```
Dim resultat As Byte
Dim a As Byte = 89
Dim b As Byte = 55
resultat = a Xor b
```

Fyll ut tabellen og finn resultatet selv.

Bruk

Dette er kanskje fancy, men hva kan det brukes til? Anta f.eks. at vi kan ha tre forskjellige feiltyper ved en inputkontroll. Vi tilordner den ene feilen feilkoden 1, den andre feilkoden 2 og den tredje feilkoden 4. Legg merke til at dette er toerpotenser.

Vi kan se for oss følgende kode i VB ("hardkodet" – ikke pent men oversiktlig):

```
Dim kontroll As Byte = 0
'feil av type 1 har inntruffet
kontroll = kontroll Or 1
'feil av type 2 inntraff ikke
'feil av type 4 har inntruffet:
kontroll = kontroll Or 4
'feil av type 4 har inntruffet igjen
kontroll = kontroll Or 4
```

La oss se hva som skjer med variabelen *kontroll* her:

kontroll = 0:	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
kontroll = kontroll Or 1:	0	0	0	0	0	0	0	1
4:	0	0	0	0	0	1	0	0
kontroll = kontroll Or 4:	0	0	0	0	0	1	0	1
4:	0	0	0	0	0	1	0	0
kontroll = kontroll Or 4:	0	0	0	0	0	1	0	1

Legg merke til at når vi bruker *Or* og ikke addisjon (pluss), så vil *kontroll* ikke bli påvirket av at feil nr 4 skjer flere ganger.

Variabelen *kontroll* ender med verdien 5. Vi ser jo lett at feiltype 1 og feiltype 4 må ha inntruffet (minst én gang), men hvordan kan programmet vårt finne det? Slik kan koden se ut:

```
If (kontroll And 1) = 1 Then MsgBox("Feil av type 1 har inntruffet")
If (kontroll And 2) = 2 Then MsgBox("Feil av type 2 har inntruffet")
If (kontroll And 4) = 4 Then MsgBox("Feil av type 4 har inntruffet")
```

Jeg utnytter *And* til å skru av alle bitene unntatt den ene som evt. angir en bestemt feiltype. F.eks. vil den første bli slik:

kontroll (=5):	0	0	0	0	0	1	0	1
1:	0	0	0	0	0	0	0	1
kontroll And 1:	0	0	0	0	0	0	0	1

Uttrykket "*And 1*" virker altså som et slags filter som kun slipper igjennom et ettall i siste bit, hvis det finnes.

Tilsvarende virker "*And 2*":

kontroll (=5):	0	0	0	0	0	1	0	1
2:	0	0	0	0	0	0	1	0
kontroll And 2:	0	0	0	0	0	0	0	0

Og "*And 4*":

kontroll (=5):	0	0	0	0	0	1	0	1
4:	0	0	0	0	0	1	0	0
kontroll And 4:	0	0	0	0	0	1	0	0

Istedenfor å filtrere med 1, 2 4 osv., kan vi anvende skift og bare "filtrere" med 1:

```
If (kontroll And 1) = 1 Then MsgBox("Feil av type 1 har inntruffet")
kontroll = kontroll >> 1
If (kontroll And 1) = 1 Then MsgBox("Feil av type 2 har inntruffet")
kontroll = kontroll >> 1
If (kontroll And 1) = 1 Then MsgBox("Feil av type 4 har inntruffet")
```

Valgene i MsgBox

Hvis du ser på hvilke valg du kan gjøre i en *MsgBox*, vil du se at mange (men ikke alle) følger dette mønsteret med kodene 1, 2 4 osv.
(http://msdn.microsoft.com/en-us/library/139z2azd%28v=vs.90%29.aspx)

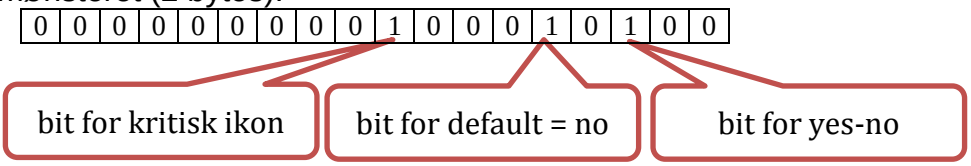
Dette eksempelet er hentet derfra:

```
Dim style = MsgBoxStyle.YesNo Or MsgBoxStyle.DefaultButton2 Or MsgBoxStyle.Critical
```

Som er det samme som å skrive

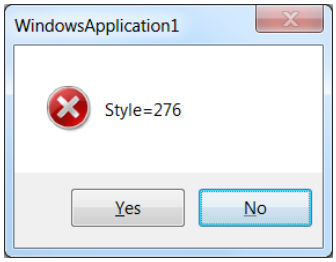
```
Dim style = 4 Or 256 Or 16
```

Det gir dette bitmønsteret (2 bytes):



Meldingsboksen med en slik *style* blir slik:

```
MsgBox("Style=" & style, style)
```



Tegntabeller

En datamaskin kan jo ikke lagre bokstaver direkte – alt i maskinen er binært. Bokstaver og alle andre tegn lagres derfor som tallkoder. Den opprinnelige tabellen var ASCII-tabellen (American Standard Code for Information Interchange), og den så slik ut:

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

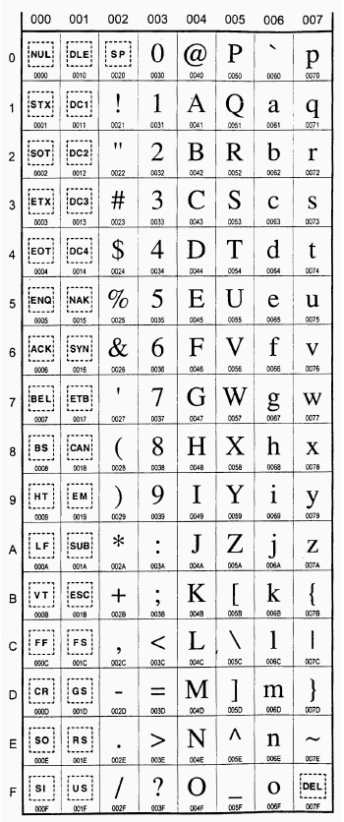
Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂*

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

ASCII-tabellen. 7 bits (+ paritetsbit)

Bokstaven *A* ble altså representert i maskinen som tallet $65_D = 41_H = 100\ 0001_B$. Det binære tallet er jo her ikke egentlig et tall men et bitmønster som representerer en bokstav. Det er bare syv bits. Den åttende biten (foran) ble brukt som kontrollsiffer[5].

I dag brukes helst *Unicode* som har 32 bits mønstre for tegnene. Da kan man kode inntil 65 536 tegn og foreløpig har det vært tilstrekkelig. Det er mulig å benytte færre bits, da *Unicode* er delt i "pages". Her gjengis *Unicode* 8 bits ("UTF-8"), side 1 (legg merke til at den er heksadesimal):



	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	STX 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	SOT 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0026	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

Unicode 8 bits side 1. (På side 2 er det flere tegn fra 0080 til 00FF)

Tabellen leses slik: *A* står i kolonnen merket 004_H og på raden merket 1_H. Disse tallene legges sammen til 41_H som er koden for *A*. Du kan se at dette er likt med den "gamle" ASCII og det er bevisst. I tillegg har side to av UTF-8 ytterligere 128 tegnkoder.

Inne i ruten sammen med *A* står tallet 0041_H. Den egentlige koden er altså fire heksadesimale siffer = 32 bits – her med de 16 første bitene satt til 0.

Fra Wikipedia:

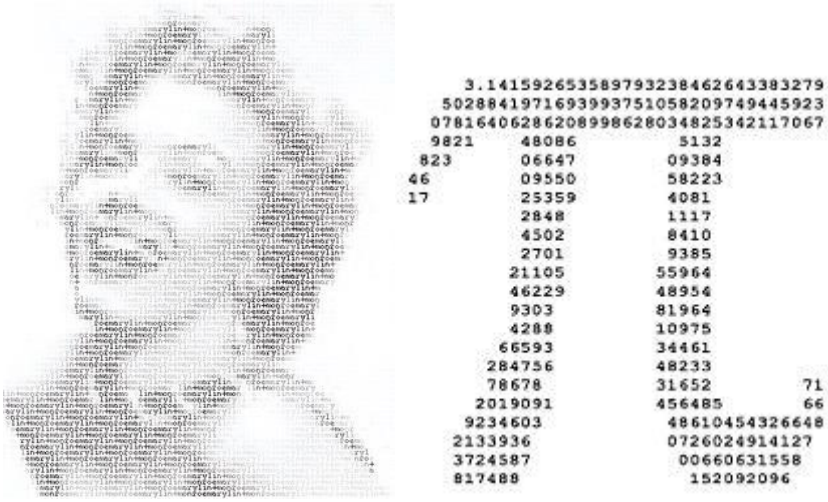
> Unicode bruker forskjellige metoder for å representere tegnene i filer. I UTF-32 brukes 32 bit per tegn for å representere hele dagens Unicode-tegnsett, mens

[5] Det var enten *oddetallspatiet* der kontrollsifferet sikret at det var et oddetall 1-ere i bitmønsteret eller *partallsparitet* der kontrollsifferet sikret et partall med 1-ere. Bitmønsteret 100 0001 ble da utvidet til 1100 0001 (tre 1-tall) med oddetallsparitet og til 0100 0001 (to 1-ere) med partallsparitet. Forskjellige systemer brukte forskjellig paritet.

i UTF-16, som brukes i nyere versjoner av Windows, deles tegnene opp i porsjoner på 16 bit. UTF-8 bruker sekvenser på 8 bit, og er gjort bakoverkompatibelt med 7-biters ASCII tegnsettet som brukes i en stor andel av verdens datasystemer, slik at det enkelt kan benyttes i en rekke operativsystemer og kommunikasjon over Internett.

Fordelen med Unicode er at det tillater flerspråklig kommunikasjon. Mens ASCII-baserte tegnsett hindrer brukeren i å skrive på flere språk innenfor ett og samme dokument, støtter Unicode de fleste språk en innenfor rimelighetens grenser kan tenkes å bruke i et datasystem. Dermed gjør Unicode det mulig å enkelt representere flerspråklig data elektronisk. For mange språk er det uten Unicode vanskelig eller umulig å finne applikasjoner som støtter det.

Noen av tegnene i tegntabellene er *kontrolltegn* som får skriveren til å gjøre noe, f.eks. å skifte til ny side, gi en lyd eller annet. Resten er *skrivbare tegn* som vil vises både på en skriver og på skjerm. Før skjermer og skrivere ble grafiske, måtte grafikk lages bare med skrivbare tegn. De sirkulerte som filer fra IT-miljø til IT-miljø og hang på de fleste "printerrom" (printerne bråkte fælt og ble derfor gjerne plassert på egne rom). Nedenfor er et par meget kjent eksempler.



Computergrafikk "i gamle dager", tegnet bare med skrivbare tegn.

Fargekoder

Heller ikke farger kan lagres direkte i maskinen. Derfor brukes fargekoder, slik at visse binære bitmønstre representerer en farge.

Mange maskiner bruker tre bytes = 24 bits til å angi fargene. Første byte er metningen av rødt (0..255), andre byte angir metningen av grønt og den tredje angir metningen av blått. Jo større tallet er i hver byte, desto mer er det av fargen. Systemet kalles derfor RGB.

Ren rød: 11111111 00000000 00000000 = FF0000
Ren blå: 00 00 FF
Gul: FF FF 00 (blander rødt og grønt)
Fiolett: FF 00 FF (blander rødt og blått)

I Visual Basic oppgis fargen ikke som RGB men som BGR med to ekstra bytes foran, f.eks. &H000000FF& som er ren rød. I HTML skriver vi f.eks. #FF0000 som er rød. For skrivere og trykksaker oppgis fargekoden heller som CMYK med metningen av Cyan, Magenta, Yellow og blacK i prosent [0..100] basert på de fargene skriveren har (i blekk eller pulver).

Jeg har laget et program som viser fargekombinasjoner, og det tenker jeg å vise.

Ekstra 1: Regning med binære tall

Du kan selvsagt regne med desimale tall og har lært utenat at 2+3=5 og kan lett regne ut at at 242+31=273 eller at 12·5=60. Når vi bruker datamaskiner er det ofte bruk for å regne med binære tall. De fire vanlige regnearter for binære tall viser jeg nedenfor.

Summering

Når vi summerer ("legger sammen") desimale tall, skriver vi gjerne slik:

```
 1 1
187
 46
233
```

De små tallene over er "i mente". 7 + 6 = 13, men bare 3-tallet får plass i siste kolonne, så 1-tallet som representerer 10 hører hjemme i kolonnen foran. Tilsvarende for 1+8+4=13 i tierkolonnen. 3-tallet får plass i tierkolonnen, mens det 1-tallet må i kolonnen foran der hundreder hører hjemme (13 tiere er etthundre og tredve).

Binært summerer vi bare enere, og det er enkelt:

```
 1 1
 101
  11
1000B
```

Øvelser

Summer de to binære tallene:

0000 0001
0000 0010

0000 0011
0000 1100

0000 0111
0000 0001

0011 0010
0100 1110

1001 0000
0001 1101

0111 1111
0000 0001

Subtraksjon
Når vi subtraherer (trekker fra) desimalt, hender det at vi må "låne". Da "låner" vi 10 fra forrige kolonne (f.eks. er 1 hundre = 10 tiere osv).

$$\begin{array}{r} {\scriptstyle 10} \\ 207 \\ -\ 37 \\ \hline 170 \end{array}$$

Binært gjør vi helt tilsvarende, men her representer 10 det binære tallet "to":

$$\begin{array}{r} {\scriptstyle 10} \\ 101 \\ -\ 11 \\ \hline 10_B \end{array}$$

Øvelser:
Subtraher (trekk fra) det andre binære tallet fra det første:
0000 1110
0000 0111

1010 1010
0001 1111

1000 0000
0000 0001

Multiplikasjon
Å multiplisere ("gange") binært er uhyre enkelt – det er bare snakk om å gange med 0 eller med 1.

Desimalt gjør vi omtrent slik:

$$\begin{array}{r} 24 \cdot 3 \\ \hline {\scriptstyle 1} \\ 72 \end{array}$$

$$\begin{array}{r} 24 \cdot 21 \\ \hline 24 \\ 480 \\ \hline 504 \end{array}$$

24·100 = 2400 (merk at når vi ganger med en tierpotens, så legger vi bare til nuller bak.)

Binært blir dette greit på helt tilsvarende måte:

$$\begin{array}{r} 101 \cdot 11 \\ \hline 101 \\ 1010 \\ \hline 1111 \end{array}$$

1011·10
0
10110
10110

(Merk at det siste regnestykket ganger med en helt toerpotens og resultatet er et samme som bare å legge til nuller bak. Det tilsvarer en venstre skiftoperasjon som datamaskiner gjør dette svært raskt.)

Øvelser
Multipliser de to binære tallene med hverandre:
101 x 110 =

1110 0101 · 10 0001 =

1010 0001 · 10 =

1010 0001 · 100 =

1010 0001 · 1000 =

Divisjon
Først minner jeg om divisjon ("deling") desimalt:
38 : 2 = 19
2↓
18
18
0 til rest

829 : 9 = 92
81↓
19
18
1 til rest

Binært ser det slik ut:
11011 : 11 = 1001
11↓
00
00↓
01
00↓
11
11
0 til rest

Hvis vi deler desimalt med hele tierpotenser, fjerner vi bare null bakerst. Tilsvarende gjør vi binært med hele toerpotenser (dette er riktig enten det er desimalt eller binært):

1100 : 100 = 11
Deling med hele toerpotenser tilsvarer høyre skiftoperasjon(er).

Øvelser:
Divider det første binære tallet med det andre:
101 1001 : 11 =

1100 0011 : 1101 =

1111 1111 : 101 =

1111 1111 : 11 0011 =

Ekstra 2: SI - Det internasjonale standardsystemet for tall og mål

SI, "Le Système International d'Unités", vedtas av den internasjonale organisasjonen CGPM, "Conférence Générale des Poids et Mesures", som er opprettet under den internasjonale avtalen "Convention du Mètre".

Prefikser

Når det blir mange siffer blir tallet uoversiktlig. Da kan det være greit å kunne forkorte skrivemåten. Min maskin har f.eks. 319 000 000 000 bytes lagringsplass. Vitenskapsfolk vil skrive det som $319 \cdot 10^9$ bytes. Det passer dårlig for menigmann. Da er det bedre med 319 gigabytes. Tallet er blitt enklere ved å sette et ord (prefiks) foran og fjerne nuller.

SI har laget et stort system for dette, hvorav mange nok vil være kjent for de fleste:

Prefiks	Symbol	Størrelse	Kommentar
lotta	L	10^{27}	**Fleip!** Engelsk (Led Zeppelin) *lotta* = mange
yotta	Y	10^{24}	Gresk/latin *octo* = 8
zetta	Z	10^{21}	Latin *septem* = 7
exa	E	10^{18}	Gresk *hex* = 6
peta	P	10^{15}	Gresk *pente* = 5
tera	T	10^{12}	Gresk *teras* = monster
giga	G	10^{9}	Gresk *gigas* = gigant
mega	M	10^{6}	Gresk *megas* = stor
kilo	k	10^{3}	Gresk *chilioi* = 1000
hekto	h	10^{2}	Gresk *hekaton* = 100
deka	da	10^{1}	Gresk *deka* = 10
deci	d	10^{-1}	Latin *decimus* = 1/10
centi	c	10^{-2}	Latin *centum* = 100
milli	m	10^{-3}	Latin *mille* = 1000
micro	µ (gresk my)	10^{-6}	latin *micro* = gresk mikros = liten
nano	n	10^{-9}	Latin *nanus* = gresk nanos = dverg
pico	p	10^{-12}	Spansk *pico* = litt eller Italiensk *piccolo* = liten
femto	f	10^{-15}	Dansk/norsk *femten*
atto	a	10^{-18}	Dansk/norsk *atten*
zepto	z	10^{-21}	Latin *septem* = 7
yocto	y	10^{-24}	Gresk/latin *octo* = 8
lotto	l	10^{-27}	**Fleip!** Engelsk lotto = svært lite, en referanse til vinnersjansen i Lotto-spillet

Merk at kg er spesiell - den er riktignok enhet, men den *egentlige* enhet er *gram. Enheten* kg (= 1000 g) brukes bare av historiske grunner. Det er følgelig *ikke* tillatt å

skrive kkg (og mene 1000 kg), man må skrive *Mg* (som tilsvarer ett tonn i vanlig norsk).

Prefikser for toer-potenser

IEC (International Electrotechnical Commission) har vedtatt prefikser for toerpotenser – ikke godkjent av SI – som følger:

Factor	Name	Symbol	Origin	Derivation
2^{10}	kibi	Ki	kilobinary: (210)1	kilo: (103)1
2^{20}	mebi	Mi	megabinary: (210)2	mega: (103)2
2^{30}	gibi	Gi	gigabinary: (210)3	giga: (103)3
2^{40}	tebi	Ti	terabinary: (210)4	tera: (103)4
2^{50}	pebi	Pi	petabinary: (210)5	peta: (103)5
2^{60}	exbi	Ei	exabinary: (210)6	exa: (103)6

Disse anbefales uttalt med første stavelse tilsvarende det metriske systemet etterfulgt av "bi" med trykk, f.eks. Ki = "kilo*bi*", og Gi = "giga*bi*".

En fin oversikt over systemet, finnes hos National Institute of Standards and Technology på http://physics.nist.gov/cuu/Units/index.html.

Ekstra 3: Grunnleggende måleenheter i SI

Bruk	Enhet	Symbol
Lengde	meter	m
Masse	kilogram	kg
Tid	sekund	s
Temperatur	Kelvin	K
Strømstyrke	Ampére	A
Mengder av substans (stoff)	mole	mol
Lysstyrke	candela	cd

Fra disse enhetene avledes definisjoner på mange andre enheter som brukes i praksis. Noen er definert *innenfor* SI, f.eks. radian (rad), herz (Hz), volt (V), Ohm (Ω), Celcius (°C), Watt (W) og mange andre.

Videre godkjenner og definerer SI bruk av tradisjonelle enheter *utenfor* SI-systemet, f.eks.

Name	**Symbol**	**Value in SI units**
minute (time)	min	1 min = 60 s
hour	h	1 h = 60 min = 3600 s
degree (angle)	°	1° = (1/180) rad
liter	L	1 L = 1 dm^3 = 10^{-3} m^3
metric ton	t	1 t = 10^3 kg
astronomical unit	ua	1 ua ≈ 1.495 98 x 10^{11} m

Oppgave til kapittel 3 (tallsystemer)

Mange studenter leser litteratur og lager notater eller merker av noe i boken. Senere repeterer de flere ganger ved å lese notatene/merkingen om igjen. Forskning viser at det gir mer læring å stille seg selv eksamensspørsmål og så prøve å besvare dem. Her skal du nettopp bruke denne, beste læringsstrategien.

1. Kjør programmet *Tallsystemer* til du klarer 20 riktige på rad med største vanskegrad (255) og med varierte oppgaver

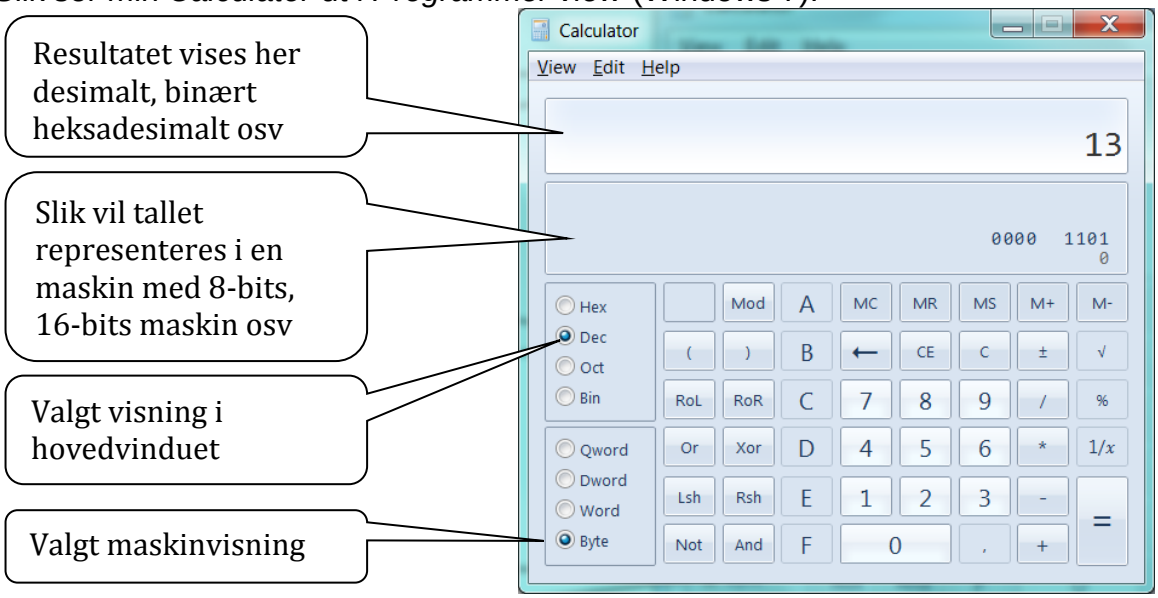
   eller

2. Lag selv disse oppgavene
   a. 5 binære tall som skal omgjøres til desimaltall og heksadesimalt tall
   b. 5 desimaltall som skal omgjøres til binærtall
   c. 5 summer av to åttesifrede, binære tall
   d. 5 differanser mellom to åttesifrede, binære tall
   e. 5 divisjoner av to åttesifrede, binære tall
      Vink: Du kan sikre at stykket "går opp" ved å gange to tall med hverandre i kalkulatoren.
   f. 5 multiplikasjoner av to åttesifrede, binære tall
   g. 5 heksadesimale tall som skal omgjøres til binært tall
   h. 5 binære tall som omgjøres til heksadesimale tall

   Bytt helst oppgavene dine med en medstudent, løs hverandres oppgaver og kontroller deretter hverandre.

   Kontroller også med en kalkulator. Bruk gjerne Microsofts *Calculator* på PC'en eller en online kalkulator f.eks.
   http://calculator.pro/decimal_to_binary-decimal_to_hex-convert_decimal.html

Slik ser min *Calculator* ut i *Programmer view* (Windows 7):



Kapittel 4 – Innledende om funksjoner

Hva er en funksjon?

Studer denne tabellen (eks. 1):

Knut	5
Kari	1
Erik	5
Olga	3
…	…

Hva betyr dette? 5-tallet er knyttet til Knut, 1-tallet til Kari osv. Hvis vi får vite at dette er navn og antall barn og at *alle navnene er forskjellige*, så kan vi *lese ut antall barn hvis vi vet navnet*. Det omvendte (navn hvis vi vet antall barn) går *ikke*.

➔ Til hvert navn er det knyttet ett, og bare ett, antall barn (men til hvert antall barn er det knyttet flere navn).

Eks. 2

tall	**kvadrat**
2	4
5	25
3	9
-2	4
…	…

Kvadrat er Tall ganget med seg selv (tall^2), altså $kvadrat = tall^2$

➔ Til hvert tall er det knyttet ett og bare ett kvadrat. Er det omvendt også?

Eks 3

tall	**dobbelt**
2	4
3	6
-2	-4
…	…

Her er $dobbelt = 2 \cdot tall$

➔ Til hvert tall er det knyttet ett og bare ett dobbelt. Her er det også omvendt da det til hvert dobbelt er knyttet ett og bare ett tall ($tall = \frac{dobbelt}{2}$).

I de to formlene er to-tallet fast (= konstant) mens både tall og dobbelt varierer.

Sett opp selv en tabell for $i = 3 \cdot s, s = (-3, -2, \ldots + 3)$
og en tabell for $s = \frac{i}{3}, i = (-9, -6, -3 \ldots + 9)$.

s	$i = 3 \cdot s$
-3	-9
-2	-6
-1	-3
0	0
1	3
2	6
3	9

i	$s = \frac{i}{3}$
-9	-3
-6	-2
-3	-1
0	0
3	1
6	2
9	3

Dette kaller vi *"å tabulere"*.

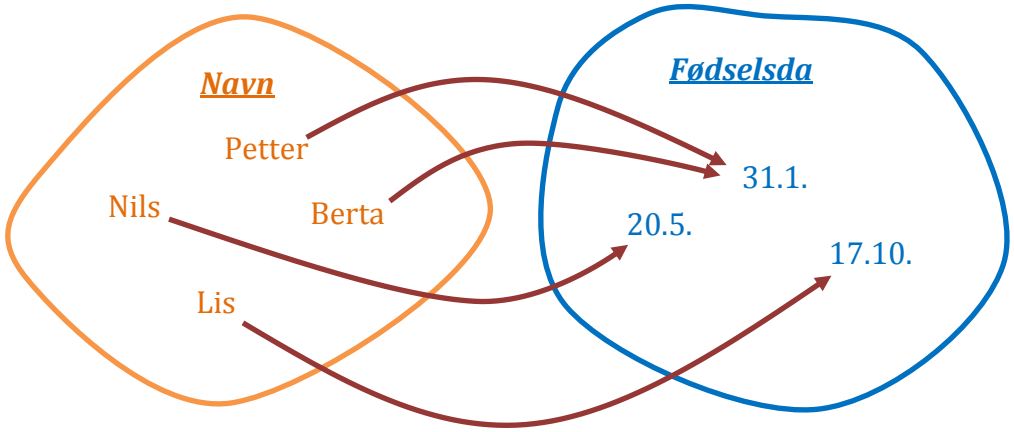
Eks 4

Navn	Fødselsdag
Petter	31.1.
Nils	20.5.
Berta	31.1.
Lisa	17.10.
…	…

Hvis alle navnene er forskjellige, utgjør de en *mengde* (husker du definisjonen du skulle pugge?) med navn. Vi ser at hvis vi vet navnet, så kan vi slå opp fødselsdagen, fordi

➔ Til hvert navn er det knyttet én og bare en fødselsdag.

Vi kan tegne det slik:



Noen fødselsdager er knytt til flere navn, men hvert navn er bare knyttet til én fødselsdag. Vi sier at *fødselsdag er en funksjon av navn* eller at *fødselsdag er funksjonelt avhengig av navn* eller at *navn determinerer (bestemmer) fødselsdag*.

Definisjon (pugges!):
En funksjon er en regel som til hvert element i én mengde A tilordner ett og bare ett element i en mengde B.

- ✓ Mengden A kalles *funksjonens definisjonsmengde*
- ✓ Mengden B kalles *funksjonens løsningsmengde*
- ✓ Funksjonen kan gis et *funksjonsnavn*, f.eks. kvadratrot, dobling, halvering, *f* og *g*
- ✓ A og B kan være samme mengde, f.eks. hvis en mengde personer er gift med andre personer i mengden.

Når vi skriver $y = f(x)$ mener vi altså at en regel, kalt *f*, tilordner ett element *y* til hvert element *x*. Vi leser det slik: "y er en funksjon av x". Det kan også leses som "x determinerer y". Det er *x* som er i definisjonsmengden (f.eks. et heltall fra -3 til +3), *f* er regelen (f.eks. "gang med tre") og *y* er i løsningsmengden (f.eks. -9, -4 ..9).

Hvis vi vil forklare hvordan regelen faktisk er, kan vi skrive f.eks.

$$y = f(x) = 5 \cdot x - 1, for\ x = (-3, -2, \dots 3).$$

Denne kan vi tabulere:

x	$\boldsymbol{y = 5 \cdot x - 1}$
-3	-16
-2	-11
-1	-6
0	-1
1	4
2	9
3	14

Siden både *y* og *x* varierer i verdi, kalles de *variable*. 5-tallet og 1-tallet er faste – de er *konstanter*.

Her er funksjonen for *omkretsen av en sirkel*:

$O = f(R) = 2 \cdot \pi \cdot R$, for R≥0

der R er sirkelens radius (avstanden fra midten av sirkelen ut til omkretsen). Her er 2 selvsagt en konstant. Det er også π som er *en konstant som har fått et navn*. Det er fordi den brukes så ofte. Som kjent er den lik 3,141592653589….[6]

Når vi har en funksjon, f.eks. den ovenfor gjengitte funksjon for sirkelens omkrets og lager en tabell av den, kalles det å *tabulere*. Hvis vi tegner den, kalles det å *grafe* (av gr. tegning).

[6] Her er en referanse til en nettside med noen flere siffer, for spesielt interesserte: http://newton.ex.ac.uk/research/qsystems/collabs/pi/

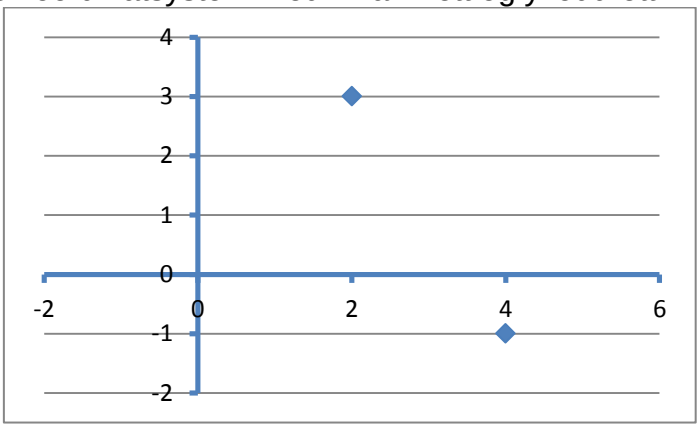
Koordinatsystemet

Når vi skal "tegne" en funksjon, gjør vi det gjerne i et plan med to akser vinkelrett på hverandre. Den ene, vannrette aksen representerer en tallinje der vi merker av verdien for den ene variabelen. Den andre, loddrette aksen utgjør en tallinje for den andre variabelen. Vanligvis bruker man den vannrette til determinanden og den loddrett til den funksjonelt avhengige variabelen. De to linje krysser hverandre der både determinanden og den avhengige er 0. Dette punktet heter *origo*.

Hvert punkt i planet representerer én verdi av hver variabel – den ene vannrett bortover og den andre lest loddrett oppover.

Eksempel

Vi tegner følgende koordinatsystem med x vannrett og y loddrett:



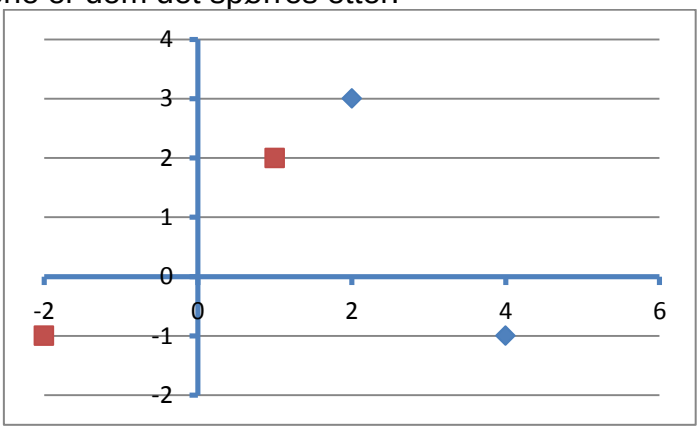
Jeg har merket av to punkter. Det ene har x-verdien 2 lest av på den vannrette aksen og y-verdien 3 lest av på den vertikale aksen. Det andre punktet har x-verdien 4 og y-verdien -1. Det første punktet skrives også slik: (x,y)=(2,3). Det andre er (x,y)=(4,-1). Hvis aksene ikke er lange nok til vårt behov, forlenger vi dem bare.

Øvelse

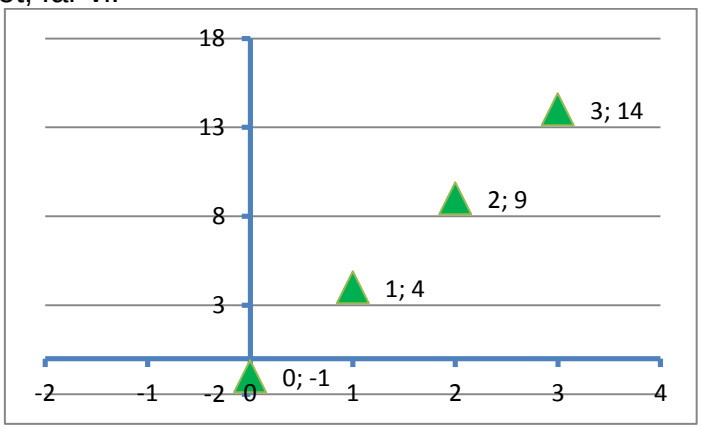
Sett inn punktene (x,y)=(1,2) og (x,y)=(-2,-1) i koordinatsystemet ovenfor.

Løsning

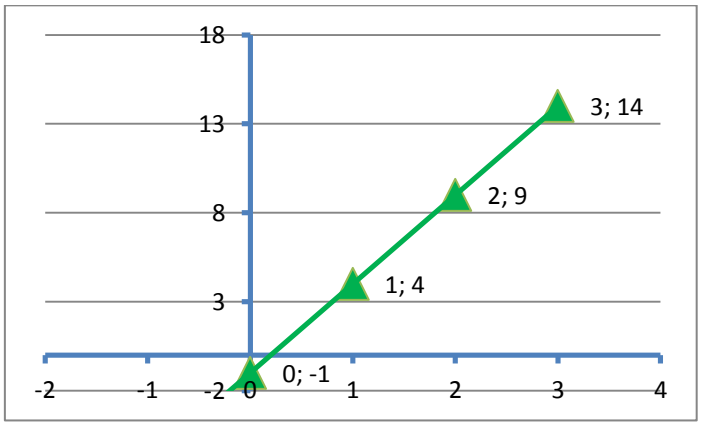
De to røde punktene er dem det spørres etter:



Ovenfor tabulerte vi funksjonen *y=f(x)=5x-1*. Når alle punktene fra tabellen plottes inn i koordinatsystemet, får vi:



Hvis vi i tillegg vet at x er et kontinuerlig desimaltall, kan vi trekke linjen mellom punktene:



Alle punkter som ligger på denne linjen være slik at *y=5x-1*. Linjen representerer da en tegning (*graf*) av funksjonen *f(x)*.

Merk deg at en funksjon har en verdi kalt *funksjonsverdien*. Verdien avhenger av verdien for en variabel som determinerer funksjonsverdien.

Eksempel
$f(x) = 2x - 1$. Da er $f(1) = 1$.

Dette skal forstås som at vi setter inn 1 for x i funksjonsregelen og beregner $f(x) = 2 \cdot 1 - 1 = 1$. Vi leser $f(1)$ som "f av 1".

Tilsvarende er $f(2) = 3$, $f(3) = 5$ osv.

Da har du repetert nok til å løse de nedenstående øvelsene.

Øvelse

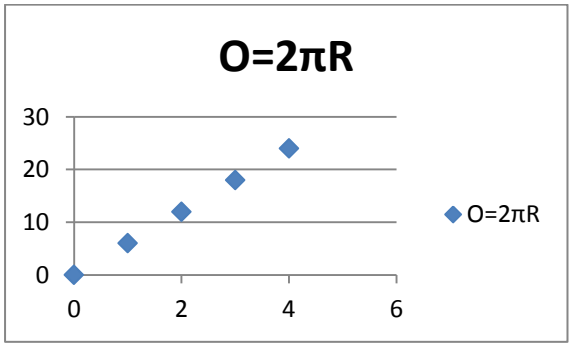
a) Tabuler sirkelens omkrets som en funksjon av radius ($O = 2\pi R$) for R lik heltall fra 0 til 4. Regn π =3 for enkelhets skyld.
b) Graf sirkelens omkrets som er en funksjon av radius for R lik *heltall* fra 0 til 4 (det vil si fem punkter).

c) Graf $O = 2\pi R$ for *alle* R fra 0 til 4 (det vil gi en rett linje).

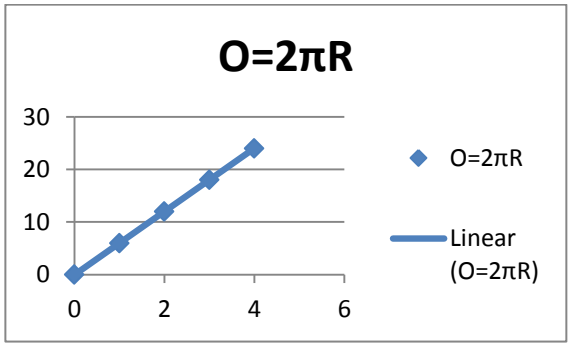
Løsning:
Tabulert:

R	$O = 2\pi R$
0	0
1	6
2	12
3	18
4	24

Grafet:



Heltallsverdier av R



Kontinuerlige verdier av R

Noen kjekke funksjoner og operatorer i Visual Basic

Val

I Visual Basic finner funksjonen *Val(x)* der *x* er en tekst (definisjonsmengden) og *Val(x)* har en tallverdi. Derfor kan vi i VB skrive

```
Dim x As Integer = Val(txtTall.Text) * 2
lblSvar.Text = x.ToString()
```

Denne henter en tekst fra txtTall (en tekstboks), gjør det om til et tall og ganger tallet med to. Så gjøres tallet om til en tekst med funksjonen toString som tilordner en tekst til et tall og setter svaret inn som en tekst i lblSvar.

Vi kan *ikke* skrive

```
Val(txtTall.Text) * 2=x
```

Det som står til venstre for tilordningstegnet (=) er det som skal få ny verdi. Det kan ikke gis ny verdi til venstresiden her.

Absolutt verdi

for alle tall *b*.
$|b|$ er den positive delen av *b*, altså *b* med positivt fortegn, dvs. *avstanden fra 0 på tallinjen*.

b	$\|b\|$
-2	2
-1,5	1,5
-1	1
0	0
1	1
2	2

…	…

I VB skriver vi

```
Math.Abs(b)
```

der *Math* er et navnerom, dvs. en samling funksjoner, konstanter og annet.

Negering

Setter vi minustegn foran et tall, bytter det fortegn. Vi skriver

$$-b$$

Det vil egentlig si at $-b$ skal være like langt fra 0 som *b* er, men på motsatt side av 0. Vi kan legge merke til at de har samme tallverdi, så $|-b| = |b|$

I VB kan vi skrive

```
b = -b
```

som betyr at *b* skal bytte fortegn (fra pluss til minus eller omvendt). I matematikken er det en umulighet, for der betyr likhetstegnet at venstresiden er lik høyresiden. Det går bare for én, eneste verdi av *b* (hvilken?).

Divisjon

I matematikken setter vi gjerne opp divisjonen som en brøk, f.eks.

$$\frac{10}{8} = \frac{5}{4}$$

som er en og en kvart.

I VB brukes skråstrek og det er to varianter:

a. Vanlig divisjon med skråstrek ("slash"):

```
b = 10 / 8
```

som gir 1,25

b. Heltallsdivisjon med bakoverskråstrek ("backslash"):

```
b = 10 \ 8
```

som gir akkurat 1, og

```
b = 10 Mod 8
```

som gir *resten* etter heltallsdivisjon og blir 2.

Merk spesielt at resten blir 2 hvis vi deler ti med åtte, men bare 1 hvis vi deler fem med fire. Her kan vi altså ikke uten videre forkorte!

Øvelse

Hva er -17 heltallsdividert med 3? Svar og rest skal angis.

Løsning

-5 med -2 til rest

Hvis resten etter en heltallsdivisjon blir 0, sies divisjonene "å gå opp". Vi kan bruke dette til å sjekke om noe er et partall. Hvis *b Mod 2* blir 0, så er b et partall, dvs. et tall delelig med 2.

```
If b Mod 2 = 0 then
```

Øvelse

Hovedsakelig er alle år som er delelig med fire skuddår. Finn ut med VB om et årstall kalt *år* i programmet er et skuddår.

Løsning
Hvis skuddår = år Mod 4 er 0 så er år et skuddår (etter denne svært forenklede regelen)

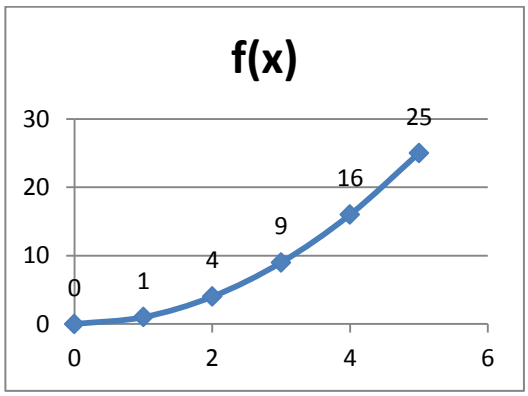
Eksponenter
Funksjoner kan ha eksponenter, f.eks.

$$y = f(x) = x^2$$

(Siden variabelen er opphøyd i annen potens, kalles slike funksjoner "annengradsfunksjoner").

La oss tabulere denne funksjonen *f* for x=0, 1, 2…5. Utfra tabellen kan vi også grafe den:

x	f(x)
0	0
1	1
2	4
3	9
4	16
5	25



Egentlig har jeg her kun grunnlag for å tegne inn punktene, for det er bare dem jeg har beregnet. I tillegg har jeg tillatt meg (som vanlig) å "interpolere" mellom punktene, idet jeg antar at de danner en glatt kurve hvis jeg hadde regnet dem ut.

Det er mange situasjoner hvor slike funksjoner er aktuelle.

I VB brukes ^ til å angi eksponenten så da skriver vi (*y* og *x* er variable) f.eks.:

```
y = x ^ 2
y = x ^ 3
```

Eksponenten kan også være negativ. Et eksempel forklarer hva det skal bety:

$$y = g(x) = x^{-3} = \frac{1}{x^3}$$

Kvadratrot
Kvadratroten til et tall *x* er det tallet som ganget med seg selv gir *x*. Det skrives $\sqrt{x}$. F.eks. er
$\sqrt{4} = 2$, $\sqrt{25} = 5$ og $\sqrt{10} = 3{,}162278$ ....

Øvelse
Hva er kvadratroten av 9?

De tallene *x* som gir et heltall for $\sqrt{x}$, kalles kvadrattallene og er 1, 4, 9,16, 25 osv., dvs. $1^2, 2^2, 3^2, 4^2$ osv.

Kvadratroten av et negativt tall er udefinert. F.eks. finnes det ikke noe tall som ganget med seg selv gir -4 (-2 ganget med seg selv gir +4). Merk også at $\sqrt{x}$ alltid har *to* løsninger: En positiv og en negativ. Derfor kan ikke kvadratrot brukes i

funksjoner – det tilordenr ikke ett og bare ett tall. Det er derfor feil å skrive $f(x) = \sqrt{x}$. Derimot kan vi skrive $f(x) = |\sqrt{x}|$ for der har regelen (høyresiden) bare en løsning for alle x≥0.

I VB finnes funksjonen Sqrt(x) som gir den positive løsningen av kvadratroten til x.

```
Math.Sqrt(5)
```

Hvis tallet det skal tas kvadratrot av er negativt, vil den gi "verdien" *NaN* som betyr "not a number".

Videre kan eksponenten i en potens være en brøk. En spesiell slik brøk er ½. Det innebærer kvadratrot:

$y = r(x) = x^{\frac{1}{2}} = \sqrt[2]{x}$ og tilsvarende for andre brøker, f.eks. $y = s(x) = x^{\frac{1}{3}} = \sqrt[3]{x}$

I VB vil følgelig alle disse gi samme resultat

```
y = x ^ (1 / 2)
y = x ^ 0.5
y = Math.Sqrt(x)
```

Heltallsdel og desimaldel

I et tall som 5,82 kalles 5 *heltallsdelen* og 0,32 *desimaldelen*. I VB finner vi heltallsdelen ved å gjøre om tallet til heltall og rund nedover. Funksjonen *Int* gir heltallsdelen, så Int(5.82) gir 5.

Ved å trekke heltallsdelen fra tallet, får vi desimaldelen.

Øvelse

a) Hvordan finner vi desimaldelen av variabelen *tall* (som er en double) i VB?
b) Hvordan kan vi runde av et tall oppover (til det minste heltall som er større enn tallet – 3,1 rundes opp til 4)?

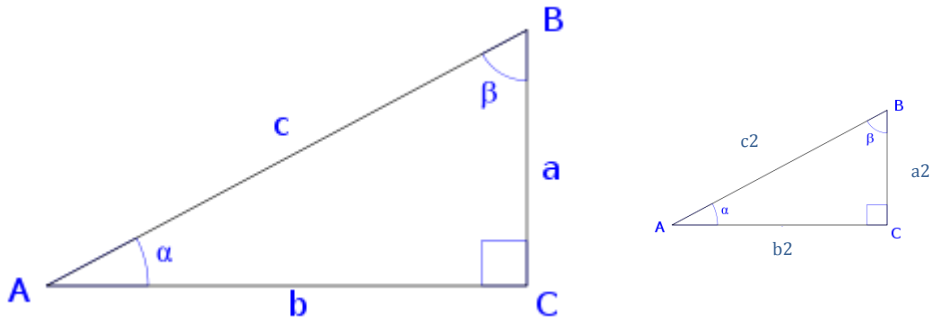
Løsning:

a) desimal = tall - Int(tall)
b) heltall = Int(desimal)+1
   eller heltall = Int(desimal + 1)

Sinus

Sinus er en av mange såkalte trigonometriske (av gr. for tre vinkler og måling) funksjoner[7]. Slike funksjoner har vært studert av matematikere i minst 1500 år. Det viser seg bl.a. at i rettvinklede trekanter (der én av vinklene er rett, dvs. 90°), er det et fast forhold mellom lengdene av sidene. Forholdet mellom dem er fast uansett trekantens størrelse hvis de har like vinkler.

[7] Wikipedia (http://no.wikipedia.org/wiki/Trigonometriske_funksjoner) her en grei oversikt over fem andre.



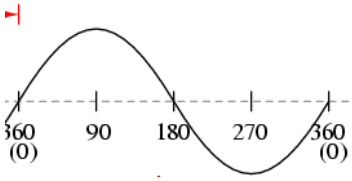
De to trekantene ovenfor er av forskjellig størrelse, allikevel er forholdet $\frac{a}{c}$ likt med forholdet $\frac{a2}{c2}$. Det faste forholdet mellom den motstående siden *a* og den lengste siden *c* sier noe om vinkelen α som er lik i begge trekantene. Forholdstallet har fått navnet *sinus til vinkelen α* og skrives *sin(α)*. Noen kjente sinusverdier:

sin(0) = 0
sin(30^o) = ½
sin(90^o)=1

Hvis vi tenker på en vanlig trekant, så kan ikke vinkelen *α* bli mer enn 90^o. Man tenker seg imidlertid at hjørnet A i trekanten ligger i Origo (nullpunktet) i et aksesystem. Da kan man dreie videre rundt og får negative sidelengder. Da oppstår også *negative sinusverdier*, f.eks.

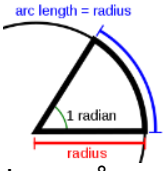
sin(300^o)=-0,866..
sin(270^o)=-1

Alle sinusverdier for vinkler fra 180 til 360^o blir negative. Slik ser det ut grafisk:

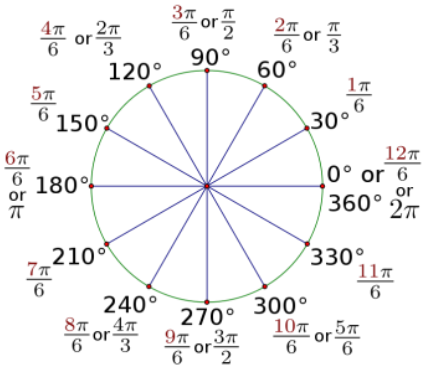


Som du ser er sinus positiv fra 0 til 180^o og deretter negativ frem til 360^o. Den har maksimumsverdien 1 ved 90^o og minimumsverdi -1 ved 270^o.

Istedenfor *grader* som er en gammel måleform for vinkler, skal man nå iflg. SI-standerden bruke *radianer*. Det er 360^o rundt sirkelen, men bare 2π radianer. Radianer defineres som avstanden langs sirkelen målt i antall radier. I figuren vises vinkelen 1 radian:



Mange synes dette er vanskelig, for de er så vant til å måle vinkler i grader. Det er imidlertid ikke vanskeligere enn at man setter av strekene rundt sirkelen annerledes, så man når 2 π (tilnærmet 6,28) istedenfor 360^o helt rundt. Man kan omregne fra grader til radianer ved å ta utgangspunkt i at $radian = \frac{\pi}{180} \cdot grader$. F.eks. beregnes 30^o slik: $radianer = \frac{\pi}{180} \cdot 30 = \frac{\pi}{6}$. Fortsetter vi rundt sirkelen får vi en sammenheng som i denne sirkelen:



Hvis du vil ha svaret i antall ganger π, blir formelen $radian = \frac{grader}{180}\ \pi$. For eksempel er da $30^o = \frac{30}{180}\ \pi = \frac{1}{6}\ \pi$ (altså samme svaret som ovenfor).

Øvelse
Bruk kalkulatoren til å finne $\sin\left(\frac{2\pi}{5}\right) = \sin(0{,}4\pi)$ og $\sin(45^o)$.

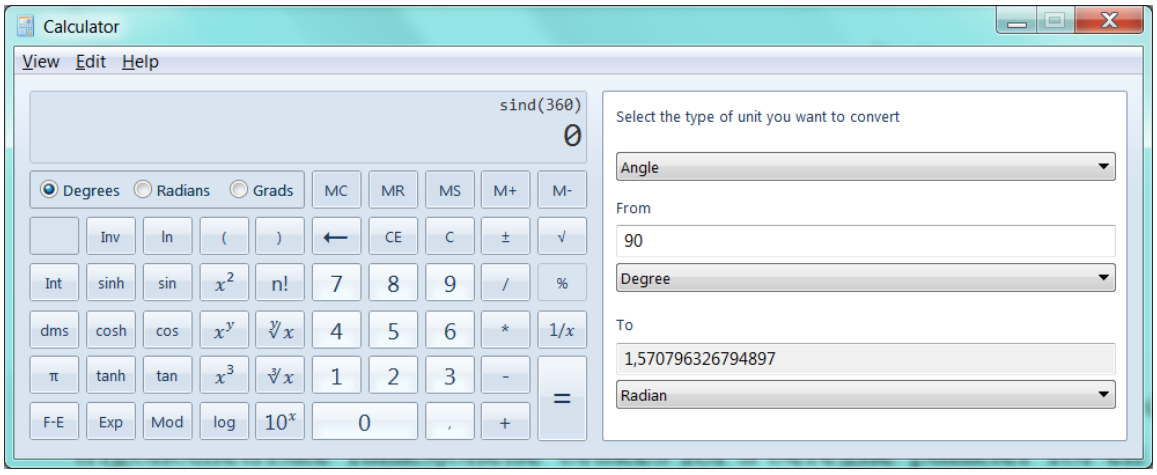
Løsning:
Omtrent 0,9511 og 0,7071.

Sinus er svært nyttig i mange sammenhenger. For oss IT-folk er det viktigste kanskje at de trigonometriske funksjonene brukes for å beregne punkter for en figur som dreies i 3D. Videre kan det være av interesse at spenningen i nettet stiger og synker med en sinusfunksjon (50 ganger pr sekund). Mange andre bølger følger også en sinusfunksjon, f.eks. lydbølger, spenninger i hjernen ("sinusbølger"), bølger på sjøen, svingninger i en gitarstreng osv.

Jeg har et program (sinus.exe) som tegner sinusfunksjonen samtidig med et punkt som beveger seg rundt en sirkel. Det tenker jeg å vise på forelesning.

Du kan også se en fin simulering nederst på siden http://www.kepi-tuebingen.de/index.php?id=196.

Kalkulatoren i Microsoft Windows kan brukes til omregning fra grader til radianer. Velg *View|Scientific* og *View|Unit conversion:*



I VB finner vi verdien π som *Math.PI*. Det er en navngitt konstant med 16 siffer. Sinusfunksjonen heter *Math.Sin(a)* der *a* skal oppgis i *radianer* (ikke i grader).

Oppgave til kapittel 4 (funksjoner)

Lag et program som er slik at du kan taste et *positivt* tall, klikke på en kommandoknapp og få regnet ut:

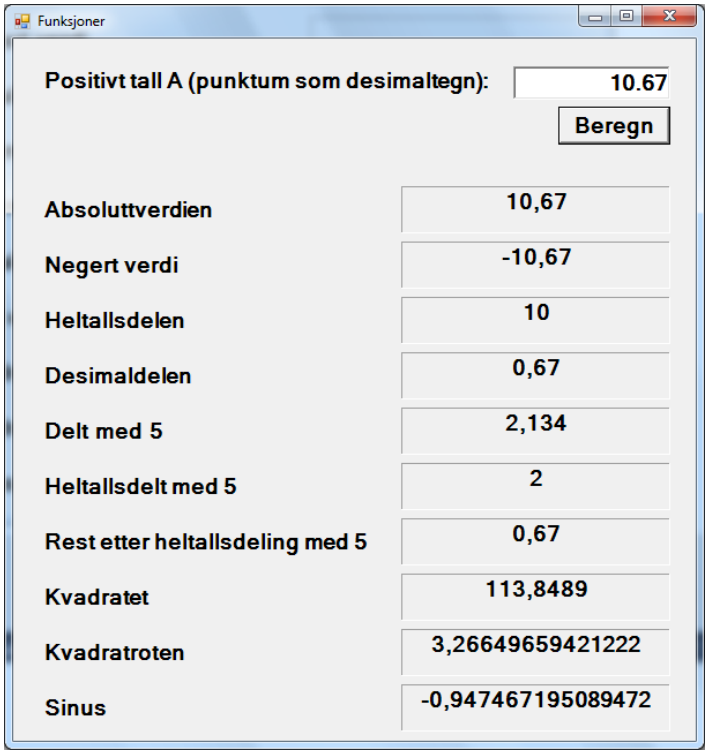
1. Absoluttverdien av tallet
2. Det negerte tallet
3. Heltallsdelen av tallet
4. Desimaldelen av tallet
5. Tallet dividert med 5
6. Tallet heltallsdividert med 5
7. Resten når tallet heltalldivideres med 5
8. Kvadratet av tallet
9. Kvadratroten av tallet
10. Sinus til tallet

Bruk programmet til å regne ut alle disse verdiene for A =

1. 4
2. 8
3. 32,777
4. 58,96
5. 104,44
6. 0
7. 4.096

Sjekk svarene dine med Microsofts Windows-kalkulator.

Skjemaet kan for eksempel se slik ut:



Kapittel 5 - Mer om funksjoner

Tilfeldige tall

Med tilfeldig mener vi uformelt noe som vi ikke kan forutsi. Mer formelt er en tilfeldig prosess definert i statistikken. Der er det en gjentatt prosess der hvert enkeltutfall er umulig å forutsi, da de ikke følger noe kjent mønster. Det er imidlertid mulig å si noe om en gruppe utfall, da utfallene følger en sannsynlighetsfordeling. Den relative andelen av hvert utfall er derfor kjent eller kan finnes. F.eks. kan vi ikke forutsi hvor mange øyne ett terningkast vil vise, men terningkastene følger en sannsynlighetsfordeling (like mange av hvert antall øyne i det lange løp) så vi kan si noe om sannsynligheten for å få f.eks. akkurat 100 "seksere" på 600 kast eller mer enn 100 "seksere" på 600 kast.

Faktisk kan det diskuteres om tilfeldighet i det hele tatt eksisterer – det kan argumenteres at hvis vi bare visste nok, kunne alt forutsies. Nøyaktig måling før og under et terningkast skulle altså gjøre det mulig å forutsi hvordan den havner. I praksis er det ikke mulig, og man benytter sannsynlighetsfordelinger isteden. De er basert på mange observasjoner og lang erfaring, men har intet bevisbart belegg.

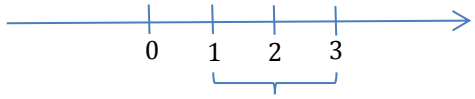
En datamaskin kan ikke gjøre tilfeldige ting – alt er styrt i algoritmer og den som kjenner algoritmen kan forutsi hvilket tall som er det neste som blir produsert. Datamaskiner er i sin natur *deterministiske*. Allikevel kan vi få maskinen til å produsere tall som er så umulige å forutsi at de fremstår som tilfeldige og som følger en sannsynlighetsfordeling. Hvis vi bruker tilfeldige tall med bare ett, binært siffer, kan vi bare få to tilfeldige tall: 0 og 1. Da kan vi *nesten* forutsi hva som blir resultatet – vi kan klare å gjette riktig annen hver gang i det lange løp. Når vi f.eks. får vite at det faktisk ble f.eks. 1, har vi fått én bits informasjon. Hvis vi bruker to bits, kan vi få 00, 01, 10 og 11. Da blir det vanskeligere å gjette. Det blir altså bedre "tilfeldighet" med mange bits. For å bli "ekte" tilfeldig, må man bruke uendelig mange bits. Det er selvsagt umulig i en datamaskin. Siden tallene ikke er "ekte tilfeldige", kalles de *pseudotilfeldige* (*pseudo random*).

Man har laget algoritmer som gir tall som er svært vanskelige å forutsi, og som tilfredsstiller alle vanlige tester på om tall er tilfeldige, og det er disse algoritmene man bruker. Algoritmene produserer ett pseudotilfeldig tall på grunnlag av et annet – gjerne det forrige pseudotilfeldige tallet. Det grunnlaget kalles da "frø" ("seed").

På Internett kan man finne virkelig tilfeldige tall, produsert på grunnlag av atmosfærisk støy av http://www.random.org/ (prøv f.eks. å kaste mynter http://www.random.org/coins/).

Intervaller

For lettere å forstå hvordan tilfeldige tall skapes i Visual Basic, er det nyttig å kjenne begrepet *intervall*. Et intervall er enkelt sagt en del av tallinjen.

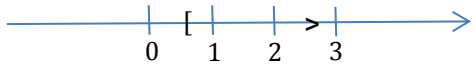


I figuren utgjør tallinjen fra 1 til 3 et intervall. Spørsmålet er om punktet 1 og punktet 3 er med eller ikke med i intervallet. For å klargjøre det, skriver vi intervallene slik:

- ✓ $[1..3]$ er intervallet fra og med 1 til og med 3, altså slik at både 1 og 3 er med i intervallet. Intervallet sies å være *lukket*.
- ✓ $<1..3>$ er intervallet fra 1 (uten 1) til 3 (uten 3), altså slik at hverken 1 eller 3 er med.
  Det er altså alle x-verdiene som er slik at 1<x<3. Intervallet sies å være *åpent*.
- ✓ $[1..3>$ er intervallet fra og med 1 (som er med) opp mot 3 (som ikke er med).
  Det er alle x-verdier som er slik at $1 \leq x < 3$. Intervallet sies å være *halvåpent*.
- ✓ $<1..3]$ er også et halvåpent intervall. Her er ikke 1 med, men 3 er det.
  $1 < x \leq 3$.

Vi kan også skrive en rekke med *heltall* på denne måten, f.eks. [1, 2, …6]. Heltallsintervaller er vanligvis lukkede, så man tar med begge ytterpunktene og noen tall mellom de to grensene for å indikere at det er heltall.

Noen ganger tegner man [og **>** direkte på tallinjen. Her er det halvåpne intervallet [½..2½> tegnet på denne måten:



Øvelse
Gitt at x er i intervallet $[1..3>$. Hvilket heltall er da $4 \cdot x$?

Løsning:
Den minst verdien x kan ha er 1. Da er $4 \cdot x = 4$. Den største verdien x kan ha er opp mot 3. Da er $4 \cdot x$ opp mot 12. Altså er $4 \cdot x$ i intervallet $[4..12>$ [8].

Pseudotilfeldige tall i Visual Basic

I Visual Basic har man funksjonen *Rnd()*[9]. Den returnerer et tall av typen *single* i intervallet [0..1> der alle tall er like sannsynlige. Verdien representeres av bare tre bytes hvilket gir $2^{64} = 16.777.216_{10}$ forskjellige verdier. Den er laget så bra at hvis man kaller den 2^{64} ganger, vil man komme innom samtlige, mulige verdier. Den er testet på alle måter av statistikere, og oppfører seg helt som forventet av et ekte, tilfeldig tall (dog med en begrensning i antall forskjellige verdier).

Det er mulig å kalle *Rnd()* med argument, men det er lite aktuelt og tas ikke her.

Eksempel 1: Kaste mynt og kron (0=Kron, 1=Mynt)
Det er to verdier og den laveste er 0.

```
x = CInt(Int(Rnd() * 2 + 0))
```

[8] Matematisk kan dette skrives slik: $x \in [1..3> \Leftrightarrow 4x \in [4..12>$ som vi leser som "x er i intervallet fra og med 1 opp mot 3 er det samme som at fire x er i intervallet fra og med 4 opp mot 12".
[9] Det finnes også en klasse *Random* som er mer fleksibel. Den ser man på i et senere kurs.

Jeg tenkte slik:

a. Rnd() går fra og med 0.0 opp mot 1.0 (single)
b. Rnd()*2 går fra og med 0.0 opp mot 2.0 (single)
c. Rnd()*2+0 går fra og med 0.0 til opp mot 2.0 (Single)
d. Int(Rnd()*2+0) går fra og med 0.0 til og med 1.0 (Single, men *Int* fjerner desimaler)
e. CInt(Int(Rnd()*2+0)) går fra og med 0 til og med 1 (Integer)

Eksempel 2: Kaste terning (1 til 6 "øyne")
Det er seks verdier og den laveste er 1

```
x = CInt(Int(Rnd() * 6 + 1))
```

Jeg tenkte slik:

a. Rnd() går fra og med 0.0 opp mot 1.0 (Single)
b. Rnd()*6 går fra og med 0.0 opp mot 6.0 (Single)
c. Rnd()*6+1 går fra og med 1.0 opp mot 7.0 (Single)
d. Int(Rnd()*6+1) går fra og med 1.0 til og med 6.0 (Single, men *Int* fjerner desimaler)
e. CInt(Int(Rnd()*6+1)) går fra og med 1 til og med 6 (Integer)

Eksempel 3: Farve i kort (1=Spar, 2=Hjerter, 3=Ruter og 4=Kløver)
Det er fire verdier og den laveste er 1.

```
x = CInt(Int(Rnd() * 4 + 1))
```

Jeg tenkte slik:

a. Rnd() går fra og med 0.0 opp mot 1.0 (Single)
b. Rdn()*4 går fra og med 0.0 opp mot 4.0 (Single)
c. Rnd()*4+1 går fra og med 1.0 opp mot 5.0 (Single)
d. Int(Rnd()*4+1) går fra og med 1.0 til og med 4.0 (Single, men *Int* fjerner desimaler)
e. CInt(Int(Rnd()*4+1)) går fra og med 1 til og med 4 (Integer)

Øvelse: Bruk Rnd() til å finne en tilfeldig kortverdi fra 2 til og med 14 (ess).

Løsning:
Det er 13 verdier og den laveste er 2.

```
x = CInt(Int(Rnd() * 13 + 2))
```

Jeg tenkte slik:

a. Rnd() går fra og med 0.0 opp mot 1.0 (Single)
b. Rdn()*13 går fra og med 0.0 opp mot 13.0 (Single)
c. Rnd()*13+2 går fra og med 2.0 til og med 15.0 (Single)
d. Int(Rnd()*13+2) går fra og med 2.0 til og med 14.0 (Single, men *Int* fjerner desimaler)
e. CInt(Int(Rnd()*13+2)) går fra og med 2 til og med 14 (Integer)

Så hva er den generelle regelen?

Til simulering av heltall som skal være like sannsynlige, kan du bruke denne formelen:

Pugges utenat:

CInt(Int(Rnd()*<antall forskjellige verdier>+<laveste verdi>))

I forelesningen tenker jeg å vise et program som simulerer myntkast, terningkast og spillkort.

Desimaltall (ikke hele tall)

Det er ikke alltid at tallen vi ønsker er heltall. Da må vi bruke en annen formel.

Til simulering av et tilfeldig tall i et intervall, kan du bruke denne formelen:

Rnd()*<intervallbredde> + <laveste verdi>

der intervallbredden er høyeste verdi minus laveste.

Eksempel: Et tilfeldig tall i intervallet [100..1000>:

```
y = Rnd() * 900 + 100
```

Note: Hvis intervallet ikke er halvåpent som her, må man gjøre noe ekstra. Det går jeg ikke inn på her.

Til simulering av andre fordelinger enn dem med lik sannsynlighet, kan vi bruke *Rnd()*, men enkelte ting må gjøres avhengig av en sannsynlighetsfordeling. Anta f.eks. at en falsk mynt har 40% sjanse for mynt og 60% sjanse for kron. Da kan vi gjøre slik:

```
If Rnd() < 0.4 Then
   MsgBox("Kron")
Else
   MsgBox("Mynt")
End If
```

Note: Pass på at *Rnd*() gir ny verdi hver gang den brukes. Derfor blir dette feil:

```
If Rnd() < 0.4 Then
   MsgBox("Kron")
ElseIf Rnd() > 0.4 Then
   MsgBox("Mynt")
End If
```

Du kan risikerer at den hverken gir kron eller mynt!

Med flere muligheter passer *Case* bedre. Her simuleres f.eks. en falsk mynt med 40% sjanse for Kron, 50% sjanse for Mynt og hele 10% sjanse for å bli stående på kant:

```
Select Case Rnd()
   Case Is < 0.4 : MsgBox("Kron")
   Case Is < 0.9 : MsgBox("Mynt")
   Case Else : MsgBox("På kant")
End Select
```

Randomize

Hver gang et program kjøres, vil det i utgangspunktet gi samme rekke med tilfeldige tall. Det er fint når programmet testes, men neppe det vi ønsker til vanlig. Da kan man kalle *randomize*. Den bruker systemklokken (som flytter frem én gang hvert millisekund) til å skape det første "frøet". Det bør man ikke gjøre inne i en løkke, da den kan gå flere ganger hvert millisekund og vil følgelig få samme frø mange ganger på rad og gi samme tall flere ganger etter hverandre. Et vanlig sted å legge *randomize* er derfor i *load*-hendelsen for skjemaet. Da blir den utført bare én gang.

Funksjoners skjæringspunkter

Når vi grafer funksjoner, vil de ofte skjære aksene i koordinatsystemet – både x- og y-aksen. Dette er spesielt interessant punkt, for ved y-aksen ser vi verdien av y når x er 0, og ved x-aksen ser vi hvilken verdi av x som gjør y=0. Det er også ofte interessant å finne skjæringspunktet mellom to funksjoner – da har de samme x- og samme y-verdi. Mange ganger kan slike punkter beregnes – vi sier at vi finner løsningen "analytisk".

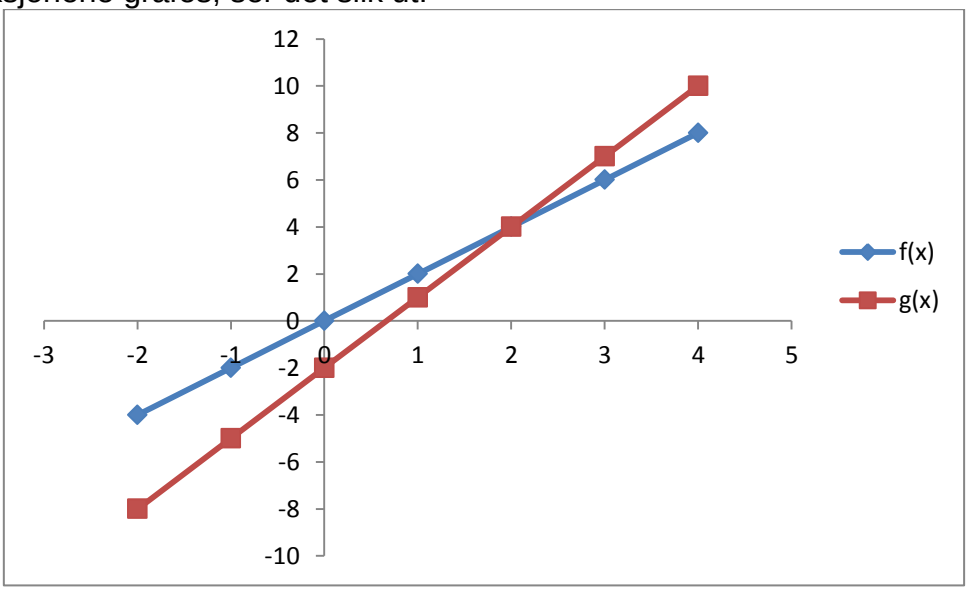
Eksempel:
Vi har to funksjoner

$$y = f(x) = 2x$$
$$y = g(x) = 3x - 2$$

Vi kan begynne med å tabulere begge funksjonene i samme tabell:

x	f(x)	g(x)
-2	-4	-8
-1	-2	-5
0	0	-2
1	2	1
2	4	4
3	6	7
4	8	10

Når funksjonene grafes, ser det slik ut:



Vi ser at begge funksjonene skjære både x (den vannrette) og y-aksen (loddrett). De skjærer også hverandre. Vi kan enkelt regne ut hvor dette skjer:

1) I skjæringen med y-aksen er x=0. Vi setter x=0 inn i funksjonene og får
   a) $y = f(0) = 2 \cdot 0 = 0$, dvs. punktet (x,y)=(0,0)
   b) $y = g(0) = 3 \cdot 0 - 2 = -2$ dvs. (x,y)=(0,−2)
2) I skjæringen med x-aksen er y=0 og vi får en enkel likning:
   a) $y = f(x) = 2x = 0 \Leftrightarrow x = 0$ dvs. (x,y)=(0,0)
   b) $y = g(x) = 3x - 2 = 0 \Leftrightarrow 3x = 2 \Leftrightarrow x = \frac{2}{3}$ dvs. (x,y)=($\frac{2}{3}$,0)
3) I skjæringen mellom de to funksjonene er f(x)=g(x) og vi får igjen en likning for å finne x:
   a) $f(x) = g(x) \Leftrightarrow 2x = 3x - 2 \Leftrightarrow -x = -2 \Leftrightarrow x = 2$
   b) Vi setter x=2 inn i én av de to funksjonene (det er det samme hvilken – de har jo samme y) og får
      $f(2) = 2 \cdot 2 = 4$ eller $g(x) = 3 \cdot 2 - 2 = 4$ dvs. (x,y)=(2,4)

For å oppsummere:

- ✓ Vi finner skjæring med y-aksen ved å sette inn x=0 i funksjonen.
- ✓ Vi finner skjæring med x-aksen ved å sette y=0
- ✓ Vi finner skjæringen mellom funksjonene ved å sette dem lik hverandre og beregne x, deretter setter vi denne x-verdien inn i én av funksjonene

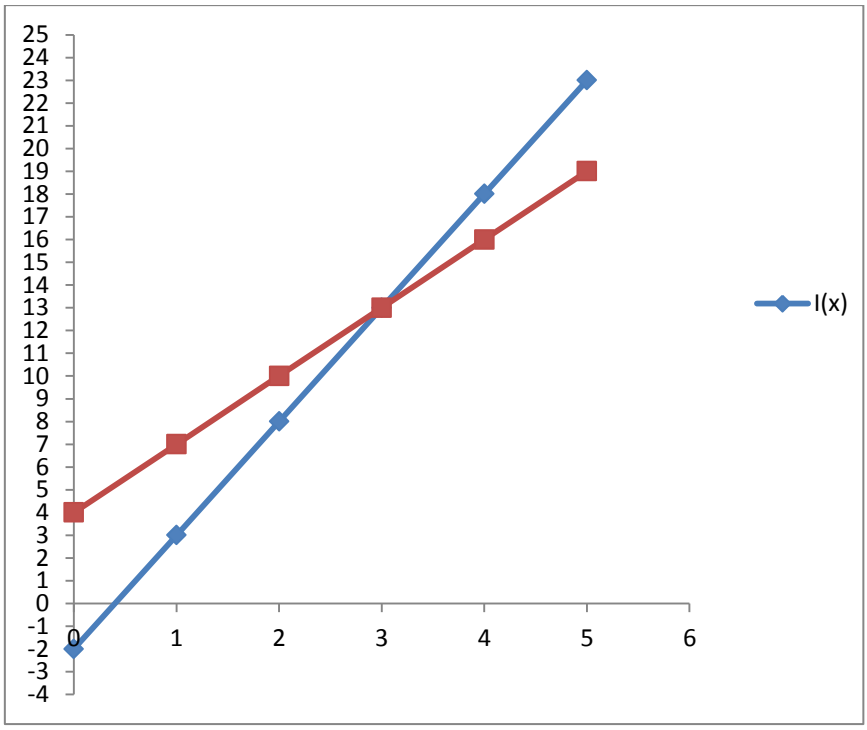
Øvelse

Gitt to funksjoner I(x)=5x-2 og K(x)=3x+4. Den første representerer inntekter (etter fradrag av salgskostnader) når vi selger x tusen stykker, den andre gjelder produksjonskostnadene. Beløpene er i hele tusen kroner.

1) Tabuler begge funksjonene
2) Graf dem fra x=0 til x=5
3) Svar på følgende spørsmål ved å lese av grafene:
   a) Hva tjener/taper du hvis du ikke selger noe i det hele tatt?
   b) Når er inntektene og kostnadene like?
4) Finn så følgende skjæringspunkter ved regning
   a) Skjæring med y-aksen for begge funksjoner
   b) Skjæring mellom funksjonene
   c) (Det har ingen hensikt å finne skjæringene med x-aksen i denne øvelsen da K(x)=0 gir negativ x, dvs. salget blir negativt og det er meningsløst)

Løsning

x	I(x)	K(x)
0	-2	4
1	3	7
2	8	10
3	13	13
4	18	16
5	23	19



Ved avlesing av grafen
Hvis vi ikke selger noe i det hele tatt, er x=0. Vi leser av kostnadene dvs. K(0)=4 og inntektene dvs. I(0)=–2. Totalt taper vi 6.

Inntekter og kostnader ser ut til å være omtrent like når salget, dvs. x, er 3 stk. Da er både kostnader og inntekter ca. 13.

Ved regning
Skjæring mellom funksjonene og y-aksen nå x=0. Vi beregner

$I(0) = 5 \cdot 0 - 2 = -2$ dvs. at I(x) skjærer y-aksen i punktet (x,y)=(0,-2)

$K(0) = 3 \cdot 0 + 4 = 4$ dvs at K(x) skjærer y-aksen i punktet (x,y)=(0,4)

Det er det samme som vi leste av grafen og det stemmer også med tabellen.

Skjæringspunktet mellom funksjonene finner vi ved å sette funksjonene lik hverandre og løse likningen:

$I(x) = K(x) \Leftrightarrow 5x - 2 = 3x + 4 \Leftrightarrow 2x = 6 \Leftrightarrow x = 3$

Det stemmer med det vi leste av. Da er kostnadene og funksjonene f.eks.

$I(3) = 5 \cdot 3 - 2 = 15 - 2 = 13$ eller

$K(3) = 3 \cdot 3 + 4 = 9 + 4 = 13$

Vi ser at de er like og det stemmer med både det vi leste av grafene og det vi tabulerte. Skjæringspunktet mellom I(x) og K(x) er altså i punktet (x,y)=(3,13).

I VB har man datatypen *Point* som har verdiene *X* og *Y*. Vi kan deklarere den og gi den verdi slik:

```
Dim skjæringspunkt As Point
skjæringspunkt.X = 2
skjæringspunkt.Y = 4
```

Dette er utenfor pensum nå, men skal brukes i et senere kurs når dere lærer å plotte (grafe) på skjermen.

Numerisk analyse med datamaskin

Ofte kan ikke skjæringspunktene beregnes slik jeg gjorde ovenfor. Da kan matematikerne få hjelp av oss programmere til å finne en *tilnærmet* løsning. Vi bruker datamaskinens rå kraft til å finne løsningen, men må akseptere at den ikke blir helt nøyaktig. Slik løsningsmetode kalles "numerisk løsning".

Eksempel:
Vi benytter samme eksempel som ved analytisk løsning og har to funksjoner

$$y = f(x) = 2x$$
$$y = g(x) = 3x - 2$$

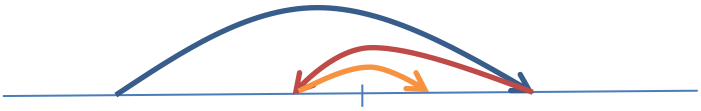
Differansen mellom dem er

$$d(x) = g(x) - f(x) = 3x - 2 - 2x = x - 2$$

Når vi skal finne skjæringspunktet mellom de to funksjonene, ønsker vi d(x)=0. Det er vårt mål, men hva må da x være?

1) Av en enkel grafing vil vi se at x må være mellom 0 og 3 et sted.
2) Vi prøver midt imellom med x=1,5 som gir $d(1{,}5) = 1{,}5 - 2 = -0{,}5$ dvs. g(x) er minst og vi må prøve lenger til høyre.
3) Prøver midt imellom 1,5 (for lite) og 3, dvs x=2,25 og beregner $d(2{,}25) = 2{,}25 - 2 = 0{,}25$. Det var altså for langt mot høyre.
4) Vi prøver nå midt imellom 1,5 (for lite) og 2,25 (for mye), dvs x=1,875 og får $d(x) = 1{,}875 - 2 = -0{,}125$ som altså er for langt til venstre.
5) Vi prøver nå mellom 1,875 (for lite) og 2,25 (for mye), dvs 2,0625 og får $d(2{,}0625) = 2{,}0625 - 2 = 0{,}0625$ som bare er bitte litt for langt mot høyre.
6) Slik kan vi fortsette og vil hele tiden nærme oss det riktige svaret. Når vi synes det er nøyaktig nok, slutter vi å regne.

Det vi gjør her er å "hoppe rundt riktig svar", men stadig nærmere:



For hvert hopp deler vi i to det området vi leter i og dette kalles derfor ofte "binærsøking". Det er en meget effektiv måte å jobbe på når vi skal lete etter en bestemt verdi i sorterte data.

Eksemplet programmert
Vi kan programmere det vi gjorde ovenfor, altså *binært søk*. Jeg viser det slik det egentlig ikke skal gjøres, med konstanter "hardkodet" for å lette forståelsen.

Her tenker jeg å vise koden og prøvekjøre programmet på forelesning.

```
For I = 1 To 1000

   'Beregn x lik midt mellom Min og Maks og d(x)
   x = (Min + Maks) / 2 'x "midt mellom"
   f = 2 * x
   g = 3 * x - 2
   d = g – f  'tilsvarer d(x) = x-2

   'Hopp ut hvis løsning er funnet
   If d = 0 Then
      MsgBox("Løsningen er funnet, x=" & x)
      Exit For 'IKKE pent, men praktisk!
   End If

   'Bytt ut enten Min eller Maks så vi kommer nærmere riktig svar
   If d > 0 Then 'x er for stor, bytt maks
      Maks = x
   Else  'x er for liten, bytt min
      Min = x
   End If

Next I
```

En annen måte å gjøre det på, er *sekvensiell søk*. Da starter vi med x=0 og så prøve x=0.001, x=0,002 osv. og hopper ut når verdien skifter fra negativ til positiv verdi for d(x)=x-2. Da er x bare *litt* for stor. Det er kanskje lettere å forstå og kan se slik ut:

```
x = 0
Do
   'Beregn neste x-verdi
   x += 0.001
   f = 2 * x
   g = 3 * x - 2
   d = g - f
Loop Until d > 0
MsgBox("Løsningen er funnet, x=" & x)
```

Dette vil imidlertid ta betydelig lengre tid. Hvis vi søker slik i store datamengder så vil det gå alt for tregt.

Hvis svaret ikke er nøyaktig nok, er det bare å søke bakover fra den siste verdien, men nå med sprang på 0,0001 til vi kommer for langt og så oppover igjen med 0,00001 osv.

Til slutt kan jeg kort nevne at man også kan søke med tilfeldige tall (her mellom 0 og 3) helt til man finner en "god nok" løsning. Dette kan gå fort eller ta meget lang tid avhengig av "flaks".

Jeg tenker å vise søking etter en løsning både med et regneark (med "målsøking) og med et program. Begge finner en løsning svært nær x=2 meget kjapt.

Øvelse
Jeg tenker på et helt tall fra 1 til 20 og du skal gjette det med færrest mulige forsøk. Du får hver gang vite om du gjetter for høyt eller for lavt. Hvordan går du frem?

Det er i alle fall ikke særlig effektivt å starte med å gjette 1, deretter 2 osv. I verste fall må du da gjette 20 ganger før du finner svaret, gjennomsnittlig (i mange spill) 10 gjetninger.

Løsning
Jeg ville startet "midt i", dvs med 10. Hvis det er for lavt, vet jeg at det må være mellom 11 og 20 og prøver midt imellom med 15 osv. Da er jeg garantert å gjette riktig på maksimalt 5 gjetninger, gjennomsnittlig (gjennom mange spill) 2 til 3. Mer effektivt går det ikke an å gjøre det.

Programmer som anvender binærsøk kan risikere å hoppe frem og tilbake rundt riktig løsning i all evighet, fordi maskinen ikke regner nøyaktig nok. Det minste, positive tall av typen *double* i VB er ca. 4,9 med komma flyttet 324 plasser til venstre, altså 0,0000….49. Det er jo svært lite, men ikke alltid lite nok. Vi kan først prøve og hvis det ikke kommer noen løsning, kan vi legge inn at maskinen skal stoppe etter en million forsøk eller så.

Istedenfor binærsøk kan vi også benytte datamaskinens regnekraft og søke sekvensielt.

Øvelse
Han far skal til Hamar med slakt, og vil benytte anledningen til å kjøpe fjærkre. Han vil ha akkurat 100 dyr og de skal koste akkurat 100 kroner. Han vil dessuten ha minst én av hvert slag. På markedet finner han haner for kr 10 pr stk, høner for kr 5 pr stk og kyllinger for 50 øre pr stk.

Hvor mange må han kjøpe av hvert slag? Bruk "brute force".

Løsning
Hvis vi ikke tenker oss om, kan vi starte med 1 hane og 1 høne og 1 kylling og kontrollere om det blir 100 stk og koste 100 kroner. Hvis ikke så prøver vi med 1 hane, 1 høne og 2 kyllinger osv. opp til 200 kyllinger. Da prøver vi 1 hane, 2 høner og 1 kylling. Slik holder vi på til en løsning er funnet.

Det blir i alt $10 \cdot 20 \cdot 200 = 40.000$ muligheter og kan ta litt tid å gjøre manuelt. Med datamaskin og "brute force" går det allikevel raskt.

En annen løsning kan være å prøve et tilfeldig antall haner (1..9) et tilfeldig antall høner og et tilfeldig antall kyllinger om og om igjen. Til slutt finner vi nok en løsning. Men dette er mye mindre effektivt enn å lete systematisk etter løsningen. Sannsynligheten for å finne riktig løsning er $\frac{1}{40\,000} \approx 0{,}000025$ for hver gang vi "gjetter".

Jeg kommer tilbake til en bedre løsning av dette problemet senere (kapittel 7).

Ekstra: Simulering (litt teori)
Simulering er opprinnelig latinsk *simuletus* av *simulare*: å kopiere, representere, late som, av lat. *similis* =lik. I denne sammenheng er det å gjøre ting ”på liksom”.

Sentralt i simulering er *modeller*. Vi lager en modell av virkeligheten, og bruker den istedenfor å gjøre det ”på ordentlig”. Grunnen til at vi gjør det kan f.eks. være at det er umulig å bruke virkeligheten (simulering av jordens skapelse), det er for dyrt, farlig, upraktisk, tidkrevende (500 generasjoner av elefanter), ulovlig (atombomber) eller vi

ønsker å si noe om fremtiden (predikering av været). Når vi lager en modell, gjør vi et *utdrag* og en *forenkling* av virkeligheten, og tar bare med de delene av virkeligheten og de egenskapene vi finner der i den grad vi mener at de er relevante for vårt formål med modellen. Noen ganger må vi ta hensyn til at modellen vil oppføre seg annerledes enn virkeligheten, f.eks. vil en liten skipsmodell i skala 1:40 ikke ha vannmotstand i samme forhold.

Hensikten er å lære noe om virkeligheten ved å studere modellen. Et alternativ til simulering er *analyse*. Da lager man en modell, og ”regner på den”. Det gir eksakte svar, men ofte er det ikke mulig eller svært vanskelig, og da er simulering en mulig løsning.

Simulering benyttes både i organisasjoner (”Management Science Techniques”) og i vitenskapene. Ved værvarsel lages f.eks. først en modell for været basert på tidspunktet for de seneste målingene. Modellen simuleres fremover i tid flere ganger med simulerte tilfeldigheter. Modellene sammenliknes så med hverandre og med været i øyeblikket og hvis de er rimelig like, kan meteorologen være ganske sikker på at modellene er riktige og velge én av dem for *yr.no*. Hvis de er ulike hverandre eller med været i øyeblikket, anses de usikre og meteorologen må velge én basert på egen erfaring. Usikkerheten angis med symboler på *yr.no* og med valg av ord i den tekstlige værmeldingen ("fare for", "mulighet for", "stort sett" o.l.).

Typer av modeller

Modeller kan være *statiske* eller *dynamiske*. Statiske modeller er faste og kan ikke ”prøvekjøres”, f.eks. en arkitekttegning. Dynamiske modeller kan ”prøvekjøres”, f.eks. en skipsmodell (kan prøve vannmotstand, stabilitet osv.).

Modeller kan være *konkrete*, f.eks. en pappmodell av et hus eller *abstrakte* som f.eks. en matematisk formel. Det er formålet og verktøyene som avgjør hva som er best. Det er f.eks. vanskelig å simulere solsystemet med en konkret modell – da er en matematisk modell bedre.

Dynamiske modeller kan være *deterministiske* eller *stokastiske*. Deterministiske modeller oppfører seg likt hver gang de prøvekjøres under se samme forhold (men det kan avhenge av forholdene hvordan oppførselen blir). F.eks. vil en pendel ha samme svingetid hver gang den prøves med samme pendellengde i samme gravitasjonsfelt – hvis vi endrer lengden eller gravitasjonen, vil svingetiden forandre seg, men hvis vi har gjort forsøket før, kan vi med sikkerhet vite hva resultatet blir (innenfor evt. målefeil). Stokastiske modeller har et element av tilfeldighet, og resultatet varierer selvom forholdene er de samme, f.eks. hvis vi simulerer terningkast. Stokastiske modeller er de mest interessante, fordi det ofte er vanskeligere å finne oppførselen ved analyse. Simulering med stokastiske modeller kalles også "Monte Carlo modeller", oppkalt etter kasinoet der.

En spesiell variant av stokastiske systemer er *kaotiske*. Det er systemer som opptrer deterministisk på kort sikt, men stokastisk på lang sikt. Været er f.eks. et slikt system.

Modeller kan være *diskrete* eller *kontinuerlige*. De diskrete modellene går i sprang fra tilstand til tilstand, slik f.eks. en familie opplever antall barn (man har ingen, ett, to osv. barn). Det er også erfaringsmessig vanskelig å være ”litt gravid”. De

kontinuerlige endrer seg kontinuerlig over tid, f.eks. alder, temperatur osv. (Vi kan simulere en kontinuerlig virkelighet med diskret modell og omvendt, men da innfører vi en feil.)

Modellene kan inkludere *mennesker*. Man setter opp en bestemt situasjon – et *scenario* – med deltakere og ”spiller” ut scenariet, slik f.eks. militære gjør når de har øvelse. Hensikten er dels at deltakerne skal lære, og dels at observatører skal lære.

Arbeidsmetode

Slik gjør vi når vi vil simulere:

1) Bygg en modell, basert på kunnskap og/eller antakelser om virkeligheten
2) Studer modellen. Hvis den er dynamisk må den prøvekjøres. Hvis den også er stokastisk, må den prøvekjøres mange ganger. Man må ta hensyn til at modellen kan oppføre seg annerledes enn virkeligheten fordi den er mindre, lettere osv.
3) Endre modellen på kontrollert måte, dvs. simuler endrede forhold og gjenta punkt 2 så mange ganger som ønskelig. Oftest kan modellen endres ved å endre visse *parametre* som beskriver kritiske forhold ved modellen.
4) Konkluder (= oppsummer læringen)

Modellspråk

Vi må da bruke ett eller annet *språk* til å bygge modellene. Språket definerer både hvilke deler som inngår og hvordan delene ”oppfører seg”. Vi kan i alle fall bygge modellene

1) fra bunnen, med et ”3GL” programmeringsspråk,
    a) vanlige ”multi purpose” språk som Visual Basic, C++ og andre
    b) spesialiserte simuleringsspråk som SIMULA
2) med modelleringsprogrammer
    a) generelle modellprogrammer som regneark (Excel)
    b) spesialprogrammer/språk for bygging av dynamiske, abstrakte modeller, f.eks. PowerSim, GPSS og SimProcess

Fixed-time eller next-event

I de spesialiserte modellprogrammene er *tiden* alltid en viktig del. Noen bruker diskret tid (PowerSim) det vil si at tiden går i sprang og modellen viser situasjonen på visse tidspunkt, gjerne med fast mellomrom. I tidsintervallet skjer det endringer i modellen, og modellen viser bare situasjonen etter at endringen er skjedd. Slike modeller kalles ”fixed-time simulation models”. Andre programmer (GPSS) har kontinuerlig tid. Én og én hendelse knyttes til et bestemt tidspunkt, og modellen stopper ved hver hendelse og viser situasjonen etter den enkelte hendelse. Slike modeller kalles ”next-event simulations”.

Fordelen med ”fixed-time” simuleringer er at modellen er enklere å bygge og raskere å kjøre. På den annen side blir det mindre data vi kan samle, fordi vi bare får data for visse forhåndsbestemte ”sjekkpunkter”. (Man kunne tenke seg at det lett løses ved å sette kort tid mellom hvert ”sjekkpunkt”, men så enkelt er det ikke.) Slike simuleringer egner seg likevel ypperlig for systemer som har en relativt jevn utvikling. Hvis systemet utvikler seg ujevnt, mister en ”fixed-time” simulering mange interessante detaljer som skjer mellom de "sjekkpunktene" man har.

Oppstart

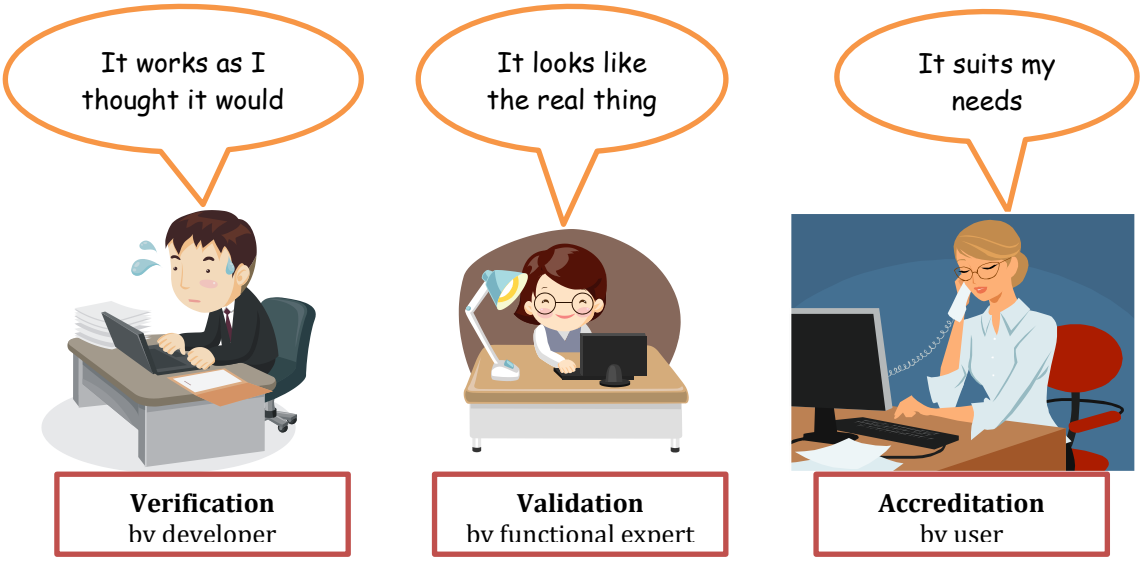
Mange systemer oppfører seg annerledes under oppstarten enn de gjør senere, når de har stabilisert seg. En kassakø vil f.eks. alltid være tom ved oppstarten om morgenen, men senere på dagen er køen kanskje (nesten) aldri tom. Derfor simuleres ofte først ”initial state” og først etter viss tid, når modellen er i ”steady state” begynner man å studere den.

Gode modeller

Sentralt for simulering er byggingen av ”gode modeller”. En god modell vil være enkel å studere og gir ”sann læring” om virkeligheten. Modellen må altså oppføre seg *analogt* med virkeligheten på de områder vi ønsker å studere. Om den ellers oppfører seg som virkeligheten, er ikke så farlig. F.eks. vil det sentrale ved en modell av de biokjemiske prosesser i menneskets fordøyelsessystem være om den kan lære oss noe om det biokjemiske. Om modellen faktisk *ser* ut som et virkelig fordøyelsessystem er da ikke så viktig. Hvis vi vil lære noe om utseende (og ikke om kjemien) blir dette naturligvis annerledes.

Et kritisk forhold ved simulering, er i hvilken grad modellen stemmer med virkeligheten. Derfor må modellen *verifiseres* (virker den som antatt)*, valideres* (stemmer den med virkeligheten og produserer riktige resultater) og *akkrediteres* (var det dette som brukerne ønsket).

En ganske vanlig form for validering, er å simulere med utgangspunkt i en kjent situasjon fra fortiden frem til en kjent situasjon senere, og sammenlikne modellen med virkeligheten. Når modellen først er verifisert, validert og akkreditert, kan den benyttes til prediksjoner.



En interessant variant for forskere, er at man bygger modellen basert på valgte forutsetninger og sammenlikner simuleringen med virkeligheten. Hvis simuleringsresultatet likner på virkeligheten, ser man på det som en støtte for at de valgte forutsetningene er riktige. Dette er en vanlig, vitenskapelig metode for astronomer, sosialøkonomer, epidemiologer, meteorologer og mange andre. Typisk er at de har mange observasjoner av virkeligheten, men ingen reell mulighet til å eksperimentere med den, slik den naturvitenskapelige metode forutsetter.

Innsamling av data
Når vi bruker datamaskin til simuleringen, vil vi la datamaskinen samle data mens simuleringen kjøres. Etterpå presenteres vi for de innsamlede dataene. Siden simuleringen gir mange tall, blir de er som regel gruppert på en eller annen måte, for eksempel. ved statistiske mål (snitt, varians, største, minste osv.).

Når vi simulerer et stokastisk system én gang, får vi bare ett, ”tilfeldig” resultat, selvom vi simulerer over lang tid og med mange hendelser. Da kan vi f.eks. få vite at maksimal ventetid i kø var 10 minutter (i denne simuleringen). Ved å simulere *mange ganger*, med samme modell og utgangspunkt, kan vi finne ut hvor ofte (=hvor stor sannsynlighet) maksimal ventetid er 10 minutter, og hvor ofte den er 9 minutter osv. Vi kan da f.eks. finne gjennomsnittlig maksimal ventetid. Det er derfor vanlig å kjøre stokastiske modeller mange ganger.

I tillegg endrer man forutsetningen for å lære noe om hvilken påvirkning de har for resultatet. Typisk vil vi i simulering av en Help Desk forsøke med forskjellig antall linjer, forskjellig antall telefonbetjenter og andre forutsetninger. Ved å knytte kostnader til forskjellige forhold, f.eks. linjer, betjenter og ventetid, kan man forsøke å finne den mest økonomiske løsningen.

Et annet viktig forhold er at man ikke kan regne med å finne den *optimale* løsningen med simulering. Man får bare sammenliknet de situasjonene man faktisk simulerer. En mulighet er å la et program gi ”tilfeldige” eller systematiske (i diskrete sprang innenfor et intervall) parametre, og la programmet samle resultatene. Det blir bedre, men allikevel får man ikke full oversikt.

Ved simulering med datamaskin, har vi to spesielle problemer. Det ene er at datamaskinen er *digital* og ikke *kontinuerlig*, slik f.eks. tiden er. Ved svært små, eller svært store verdier, kan dette gi problemer. Som regel kan man komme rundt det ved å skalere modellen. Det andre problemet oppstår ved stokastiske simuleringer, fordi de tallene som genereres ikke er *ekte* tilfeldige, bare *pseudo* tilfeldige. Videre er det slik at når man genererer mange slike tall, så vil tallrekken før eller siden gjenta seg. Hvis man f.eks. simulerer veksten av trær, så vil man altså før eller siden få et tre som er nøyaktig likt ett man har hatt før. Det er derfor noe begrenset hvor mange eller hvor lenge man kan simulere. Resultatet blir jo ikke bedre når simuleringen begynner å gjenta seg. I GPSS genereres f.eks. 2 147 483 646 (vel 2 milliarder) forskjellige tall før tallrekken gjentar seg. For å avhjelpe dette, har GPSS også *mange* generatorer for pesudotilfeldige tall, og de er uavhengige av hverandre.

Et annet forhold man bør være oppmerksom på, er at jo flere hendelser man simulerer – altså jo lengre tid man simulerer for – desto større er sjansen for at et ekstremt resultat kommer med (”freak result”). Man bør vurdere om det er fornuftig. Er det f.eks. realistisk å simulere en kassakø gjennom femten år? Da kan maksimal ventetid i kø f.eks. bli 20 timer, selvom det kanskje bare skjedde én gang på 15 år. Dessuten er modellen antakelig feil – ingen gidder vel å vente tyve timer i en kassakø☺? Andre ganger er vi spesielt interessert i nettopp disse ekstreme verdiene, f.eks. når man simulerer bølgehøyder i Nordsjøen for å vurdere konstruksjonen av en oljerigg. Da er "hundreårsbølgen" den største bølgen som kan forventes med en viss sannsynlighet i en hundreårsperiode.

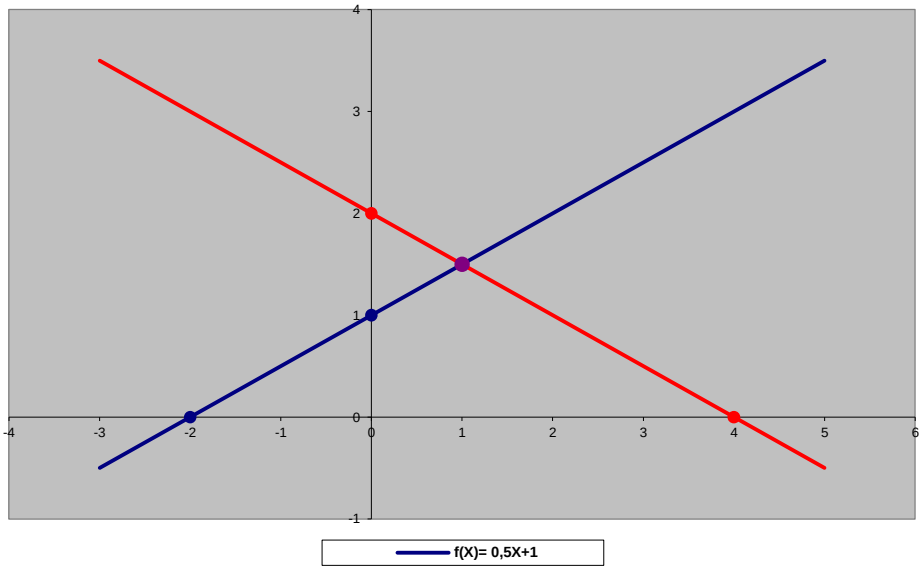
Oppgave til kapittel 5 (mer om funksjoner)

Gitt to funksjoner *f* og *g*, slik at

1) $Y = f(X) = \frac{1}{2}X + 1$

2) $Y = g(X) = -\frac{1}{2}X + 2$

Når de to funksjonene grafes i samme koordinatsystem, ser det omtrent slik ut:



Vi er interessert i å finne alle fem skjæringspunktene (angitt med prikker i grafen).

Oppgave A

Beregn alle skjæringspunktene (matematisk analyse).

Oppgave B

Beregn skjæringspunktene med Y-aksen med et VB-program. [**Vink:** Her er det ikke bruk for iterasjoner – svarene kan beregnes direkte som f(0) og g(0).] Vis gjerne svaret i en meldingsboks.

Oppgave C

Finn skjæringspunktene med X-aksen numerisk med et VB-program. Ta utgangspunkt i at alle punktene ligger et sted mellom –5 og +5. Du kan regne med at svaret finnes etter 100 iterasjoner [**Vink:** Du kan altså bruke en for-løkke.] Vis gjerne svaret i en meldingsboks.

Oppgave D

Utvid programmet så det også finner skjæringspunktet mellom de to funksjonene numerisk. Ta igjen utgangspunkt i at alle punktene ligger et sted mellom –5 og +5, bruk igjen 100 iterasjoner og vis gjerne svaret i en meldingsboks.

Oppgave E – Utfordring, veiledes ikke

Gjør om programmet så det viser svaret med en gang det er funnet. [**Vink:** Bruk en annen iterasjon enn for-løkke.] Tenk også gjennom hvordan du kan løse problemet hvis maskinen ikke klarer å finne et eksakt skjæringspunkt, slik at iterasjonen blir gående ”evig”.

Kapittel 6 – Rekker og funksjoner av flere variable

Dette er tredje gang jeg tar opp funksjoner. Det skyldes at funksjoner er svært anvendt i IT og det er helt avgjørende å forstå hva det er og hvordan det virker.

Rekker og rekursjon

Rekker er en serie med tall, av noen også kalt *tallfølger*. Tallene følger et system som gjør det mulig å finne alle tallene i rekken, kalt rekkens *ledd*. *Rekursjon* har vi når regelen som tilordner et tall i rekken er basert på et/flere tall tidligere i rekken.

Eksempel 1

Regel 1: Det første tallet er 1 dvs $x_1=1$

Regel 2: Alle andre tall i rekken er det dobbelte av det forrige tallet i rekken dvs. $x_i=2x_{i-1}$ for $i>1$.

Som vi ser, utgjør de to reglene til sammen en *funksjon* der definisjonsmengden og løsningsmengden er fra samme mengde (til hvert tall i rekken tilordnes ett og bare ett tall i den samme rekken). Dette kalles *rekursiv funksjon*. Hvis slike funksjoner ikke skal gi uendelig rekursjon er det alltid minst en regel som ikke tilordner et annet ledd i rekken, men et tall (her regel 1). Slike regler kalles "enkle" i motsetning til de andre som er "rekursive".

Vi kan sette opp rekken slik:

x_1	x_2	x_3	x_4	x_5	osv
1	2	4	8	16	…

Tallene i rekken kalles *ledd* og indeksene til hver x (i-en i x_i) angir hvilket ledd det er. Det første leddet er definert ved regel 1, alle de andre ved regel 2.

Denne rekken viser f.eks. antall celler på tidspunkt 1, 2 osv. når hver celle deler seg i to mellom hvert tidspunkt.

Legg merke til at for å finne ledd x_5 må vi først vite hva x_4 er, men da må vi først vite hva x_3 er osv. til vi treffer på en enkel regel, her $x_1=1$. Det er dette som gir rekursjonen.

Det er også mulig å finne x_5 utfra følgende regel: $x_i=x^{i-1}$, altså "x ganget med seg selv *i-1* ganger". Da gjør vi om funksjonen fra å være rekursiv til å være iterativ (som vi i VB skriver med for-løkker o.l.). Det er typisk at rekursive funksjoner kan omgjøres til iterative og omvendt. Faktisk har vi hatt mange programmeringsspråk som bare hadde iterasjon og mange med bare rekursjon og da måtte programmereren ofte gjøre om. Nå har alle(?) begge deler.

Allikevel har rekursjon stor betydning, da den ofte modellerer en utvikling bedre. Det er da lettere å komme på de rekursive reglene enn den iterative. Det kan også være svært tungvint å programmere løsningen iterativt. Videre opptrer ofte utilsiktet rekursjon i hendelsesorientert programmering, når én hendelse trigger en annen som igjen trigger den første. Resultatet er uventet og feilen ofte vanskelig å finne, men forståelse for fenomenet hjelper. Det er derfor svært vanlig at høyskoler/universiteter underviser i rekursjon både i matematikk og programmering.

Eksempel 2

1. $F_1=1$
2. $F_2=1$
3. $F_i=F_{i-2}+F_{i-1}$ for i>2

Dette er et eksempel på en rekursiv funksjon med *tre* regler. De to første er enkle, den tredje er rekursiv og baserer seg på de to foregående leddene.

Øvelse
Lag en tabell for de første seks leddene, F_1 til F_6.

Løsning

F_1	F_2	F_3	F_4	F_5	F_6
Regel 1	Regel 2	Regel 3	Regel 3	Regel 3	Regel 3
1	1	1+1=2	1+2=3	2+3=5	3+5=8

Dette er Fabonacci-tallene som har vært kjent svært lenge og dukker opp på de mest overraskende steder i naturen og matematikken[10].

Eksempel 3

1. $x_1 = \frac{1}{2}$
2. $x_i = \frac{1}{2}x_{i-1}$ for x>1

x_1	x_2	x_3	x_4	x_5
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$

Du har nå sikker kjent igjen "toerpotensene" under brøkstreken. Da ser du kanskje at reglene kan omskrives til følgende iterasjon: $x_i = (\frac{1}{2})^i$.

Noen ganger er det interessant å summere alle leddene. Det kan her skrives slik:
$SUM = \sum_{i=1}^{\infty}(\frac{1}{2})^i$
Vi skal altså summere alle leddene i det det uendelige. Er det mulig å si hva summen blir, eller blir den uendelig stor?

Her kan vi bruke datamaskinen og generere ledd for ledd og legge det til en sum og se hva vi får. *Jeg tenker å vise dette programmert i forelesningen.*

Det er også mulig å tenke grafisk her:

Vi starter med et linjestykke som er 1 langt. Vi deler det i to og får ½. Til dette legger vi ¼ som er halvveis videre til 1, deretter 1/8 som er halvveis videre derfra til 1 osv. Jo flere ledd vi legger til, desto nærmere kommer vi 1. Vi kan altså se at summen *nærmer seg 1* når *i går mot uendelig*. Matematikerne vil si at summen *konvergerer* mot 1.

[10] Du kan se noen eksempler på http://en.wikipedia.org/wiki/Fibonacci_number.

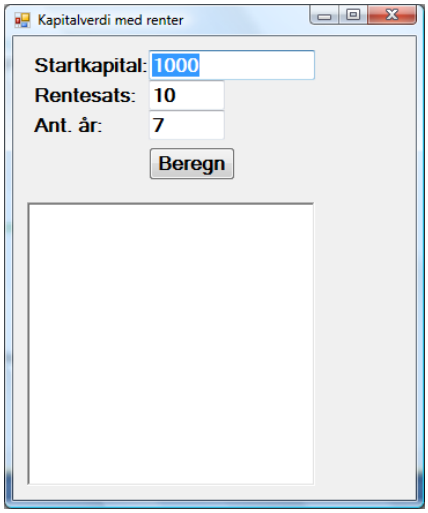
Summerer vi derimot leddene i Fibonaccirekken ovenfor, vil svaret bare vokse og vokse jo flere ledd vi legger til. Summen vokser over alle grenser, eller som matematikerne vil si, summen *divergerer*.

Eksempel 4
Regel 1: $K_0 = \text{Startkapital}$
Regel 2: $K_t = K_{t-1} + K_{t-1} \cdot \% \Longrightarrow K_{t-1}\left(1 + \frac{p}{100}\right)$
Anta at p% = 10% og Startkapital = 1000. Da kan regel 2 skrives slik:
Regel 2: $K_t = 1{,}1 \cdot K_{t-1}$

Hva er kapitalen vokst til etter fire år?

Løsning

t:	0	1	2	3	4
Regel:	1	2	2	2	2
K_t:	1000	$K_2 = 1{,}1 \cdot K_{2-1}$ $= 1{,}1 \cdot K_1$ $= 1{,}1 \cdot 1000$ $= 1100$	$K_2 = 1{,}1 \cdot K_{2-1}$ $= 1{,}1 \cdot K_1$ $= 1{,}1 \cdot 1100$ $= 1210$	$K_3 = 1{,}1 \cdot K_{3-1}$ $= 1{,}1 \cdot K_2$ $= 1{,}1 \cdot 1210$ $= 1331$	$K_4 = 1{,}1 \cdot K_{4-1}$ $= 1{,}1 \cdot K_3$ $= 1{,}1 \cdot 1331$ $= 1464{,}10$



Dette tenker jeg å vise i forelesningen ferdig programmert som et eksempel.

Rekursjon i programmering

Note: Rekursjon i form av tallrekker er pensum (beregningene på papir) men ikke rekursiv programmering er ikke pensum. Det blir pensum i senere kurs og dukker opp flere ganger i løpet av studiet.

Eksempel
Jeg har smerter og tar 500 mg smertestillende. For hver time som går, blir 1/3-del av det som er igjen i kroppen "brukt opp". Når det er gått fire timer kjenner jeg smertene igjen. Hvor stor dose er det da igjen i kroppen?

Løsning
Jeg setter opp reglene for rekken *rekursivt*:

1. Ved tidspunkt 0 (start) har kroppen 500 mg smertestillende medisin
   dvs. M_0=500
2. Ved tidspunkt *t* (etter *t* timer) er det igjen 2/3-deler av det som var igjen ved forrige tidspunkt
   dvs $M_t = M_{t-1} \cdot \frac{2}{3}$

Rekken ser slik ut:

M_0	M_1	M_2	M_3	M_4
500	333,33	222,22	148,15	98,77

Iterativt kan det utrykkes slik:

$M_t = 500 \cdot \left(\frac{2}{3}\right)^t$ altså 500 ganget med 2/3 *t* ganger.

Jeg tenker å vise dette programmert i forelesningen.

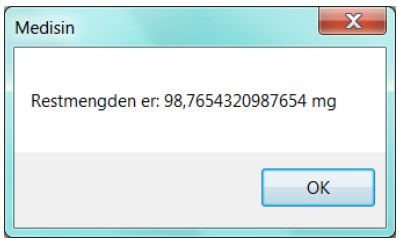
Jeg programmerer dette først iterativt – alt med "hardkoding" av tall for forståelsens skyld:

```
Private Sub butIterativt_Click(ByVal sender As System.Object,
                               ByVal e As System.EventArgs) Handles butIterativt.Click
    'Beregner medisinmengden iterativt
    Dim restmengde As Double = 500
    For t As Integer = 1 To 4  'ganger med 2/3 fire ganger
        restmengde = restmengde * 2 / 3 ' merk restmengde på begge sider av tilordning, regel 2
    Next
    MsgBox("Restmengden er: " & restmengde & " mg")
End Sub
```

Deretter viser jeg en rekursiv løsning:

```
Private Sub butRekursivt_Click(ByVal sender As System.Object,
                               ByVal e As System.EventArgs) Handles butRekursivt.Click
    'Beregner restmengden rekursivt
    Dim restmengde As Double
    restmengde = rest(4) 'rest etter fire timer
    MsgBox("Restmengden er: " & restmengde & " mg")
End Sub

Function rest(ByVal t As Integer) As Double
    If t = 0 Then Return 500 'enkel regel
    Return rest(t - 1) * 2 / 3 'rekursiv regel
End Function
```

Selv synes jeg de rekursive reglene er lettere å finne utfra beskrivelsen av hva som skjer. Det er jo ikke sikkert at du er enig for det krever litt tilvenning. For meg er det slik at når jeg først har de rekursive reglene, er funksjonen *rest* ganske enkel å skrive, for den følger reglene direkte. Begge metodene vil vise det samme svaret, nemlig



Foreløpig vil dere altså bare bli bedt om å sette opp leddene i rekken på papir, ikke programmere den.

"Smartinger" vil kanskje også se at *butRekursivt* kunne sett bare slik ut:

```
Private Sub butRekursivt_Click(ByVal sender As System.Object,
                               ByVal e As System.EventArgs) Handles butRekursivt.Click
    'Beregner restmengden rekursivt
    MsgBox("Restmengen er: " & rest(4) & " mg")
End Sub
```

Funksjonen *rest* er det vanskelig å forenkle.

Funksjoner av flere variable

Hvis A er *funksjonelt avhengig* av B, sies B å være *determinand* for A. Når man lager databaser er det av forskjellige grunner viktig å lete etter slike determinander.

Eksempel 1 – Innholdet i en databasetabell er slik:

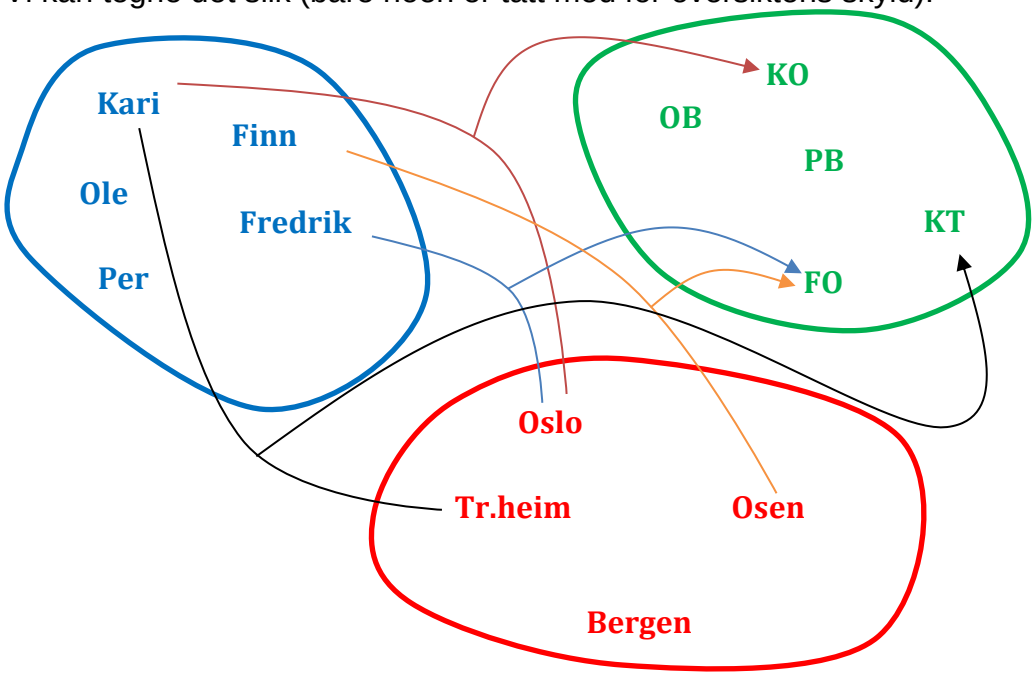
navn	by	kode
Kari	Oslo	KO
Ole	Bergen	OB
Per	Bergen	PB
Kari	Tr.heim	KT
Fredrik	Oslo	FO
Finn	Osen	FO

Vi ser at flere navn er like, og flere byer er like, og flere har samme kode (hva nå det måtte være). Ingen av dem er følgelig determindand. Hvis vi ser litt nøyere etter, vil vi se at kombinasjonen *navn+by* er unik. Hvis dette alltid holder, så er *navn+by* determinand for *kode*. *kode* er da funksjonelt avhengig av *navn+by*:

kode=f(navn,by)

Det skal bety at *for hvert par av navn og by, finnes det én og bare en kode*.

Vi kan tegne det slik (bare noen er tatt med for oversiktens skyld):



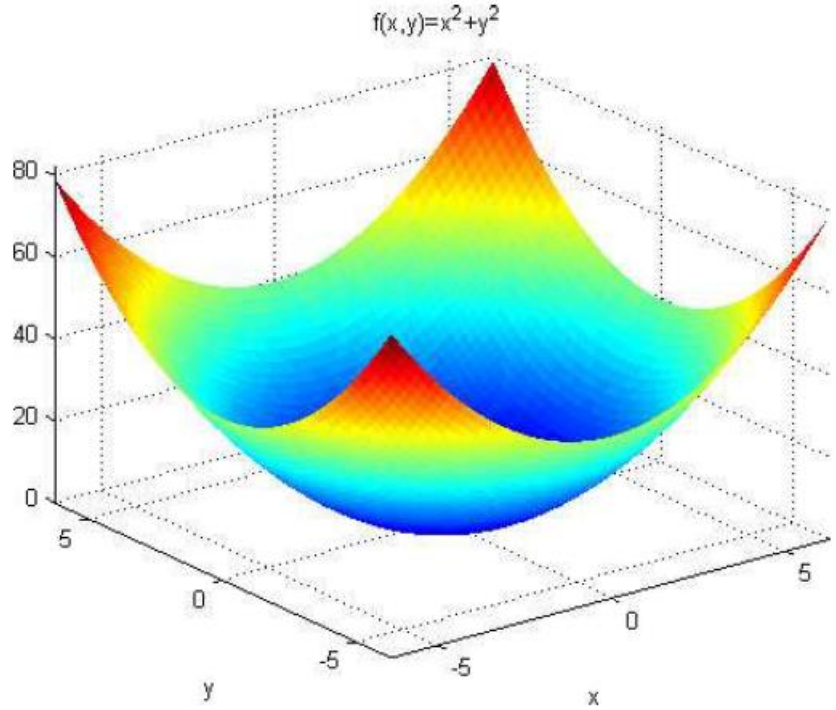
Vi ser at Kari både kan kombineres med Oslo og da er koden KO, og med Tr.heim og da er koden KT. Vi ser også at både Fredrik + Oslo og Finn+Osen gir koden FO. Både navn og by kan inngåe i flere kombinasjoner, og det kan være flere kombinasjoner som gir samme kode. Hver enkelt kombinasjon gir imidlertid bare én kode.

Eksempel 2
Gitt funksjonen $z = x^2 + y^2$ der $x \in [-5..+5]$ og $y \in [-5..+5]$.

Vi kan tabulere denne for noen kombinasjoner av verdier av x og y:

x	**y**	**z**
-5	-5	50
-5	0	25
-5	+5	50
0	-5	25
0	0	0
0	5	25
5	-5	50
5	0	25
5	5	50

Denne er ikke enkelt å grafe, men den vil se omtrent slik ut:



Det er z-verdien som går oppover, mens x- og y er gjengitt som dybde/bredde (og y-aksen er litt uvant snudd).

Når vi har to "frie" variable som til sammen er determinand, så blir det tredimensjonalt.

Med flere variable blir det helt håpløst å tegne, men det er ikke uvanlig i databaser å ha determinander med både tre og fire variable. For å finne en bestemt eiendom kreves f.eks. både kommunenummer, gårdsnummer og bruksnummer.

Øvelse:
Tabuler funksjonen $z = f(x, y) = x + y$ for $x \in [1..4]$ og $y \in [5..6]$. Tabuler bare heltall.

Forsøk å grafe den også.

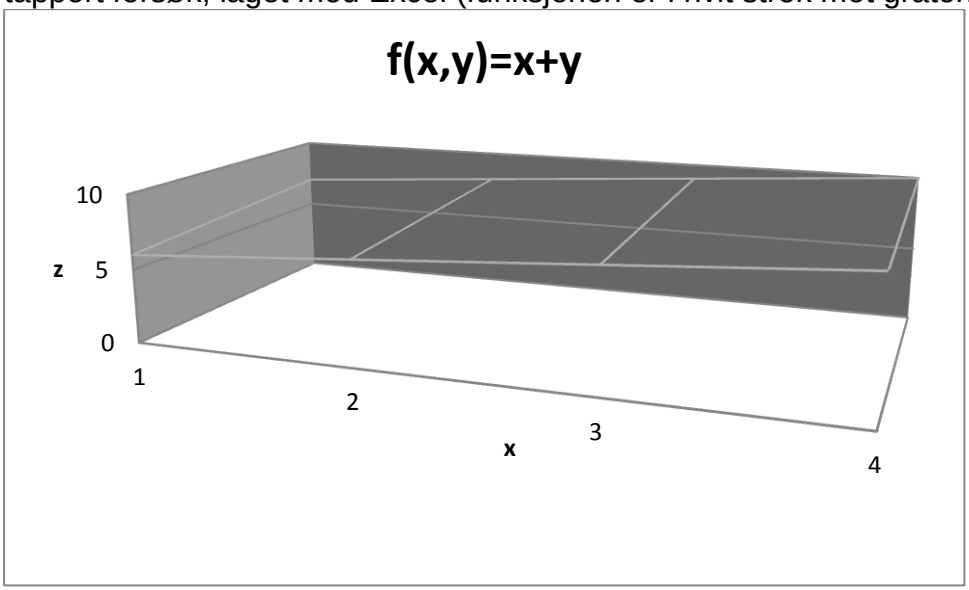
Løsning:

x	y	z
1	5	6
1	6	7
2	5	7
2	6	8
3	5	8
3	6	9
4	5	9
4	6	10

Alternativ:

	y	
x	**5**	**6**
1	6	7
2	7	8
3	8	9
4	9	10

Det er ikke lett å tegne denne heller men den ser ut som et skråplan. Her er et tappert forsøk, laget med Excel (funksjonen er i hvit strek mot gråtonede "vegger"):



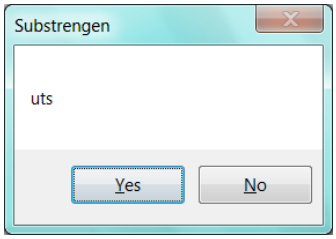
I VB finner vi ofte funksjoner av både to og flere variable. F.eks.

```
Dim s As String = "Knutsen"
s = s.Substring(2, 3)
Dim svar As Integer
svar = MsgBox(s, MsgBoxStyle.YesNo, "Substrengen")
```

Her returnerer *substring()* en streng basert på to argumenter (to heltall) og i tillegg er også strengen *s* en variabel her da den også påvirker resultatet – med en annen streng blir resultatet annerledes. Videre vil MsgBox returnere et heltall, basert på

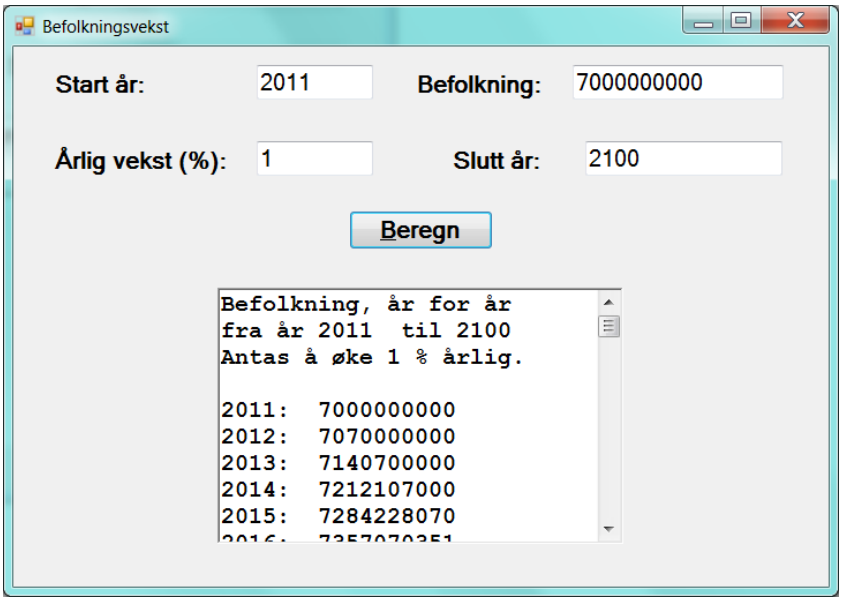
brukerens svar, og trenger tre argumenter for å vise meldingsboksen riktig (flere av dem er riktignok *optional*).

Meldingsboksen vil se slik ut:



Oppgave til kapittel 6 (rekker)

Det er vanlig å regne med at befolkningsveksten er ganske fast over noen år. I 2011 passerte vi ca 7 milliarder (7 000 000 000) og den vokser nå med ca 1 % hvert år. I dette programmet skal du beregne befolkningsstørrelse for et antall år. Skjemaet kan se slik ut:



Som du ser, skal brukeren fylle inn startåret, befolkningsstørrelse i startåret og den årlige veksten (i prosent), samt det siste året befolkningen skal beregnes for. Når brukeren klikker ”Beregn”, skal befolkningen beregnes år for år, og resultatet vises, fra og med startåret til og med sluttåret som brukeren har oppgitt.

Vink

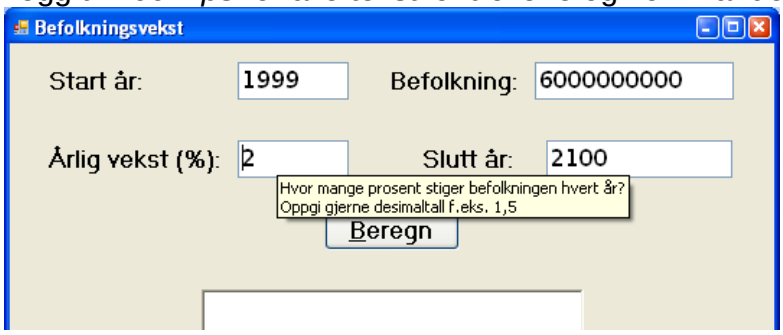
- ✓ Kontrollen der resultatet vises, er en *RichTextBox*. Hvis den heter f.eks. *rtfBefolkning*, kan du tømme den for tekst med *rtfBefolkning.Clear()* og du legger til tekst i den ved å endre *rtfBefolkning.Text*. Husk også at ny linje heter *vbNewLine* eller *vbCrLf*.
- ✓ Brukeren oppgir veksten i prosent. Du gjør om ved å dele tallet med 100, f.eks.
  vekst = vekst / 100
- ✓ Anta at du har kalt befolkningsstørrelsen for *befolkning* (*Long*) og veksten for *vekst* (*double*). Da øker du befolkningen fra ett år til et annet ved å skrive
  befolkning += befolkning * vekst
  I tillegg må du kanskje foreta noen konverteringer.

Utfordringer (veiledes vanligvis ikke)

1) Sett på nøyaktig typekontroll ved å skrive *Option Strict On* helt i starten av programmet. Da vil du antakelig få feilmeldinger, fordi du har vært litt slapp med typekonvertering. Gjør om slik at alle feilene rettes.

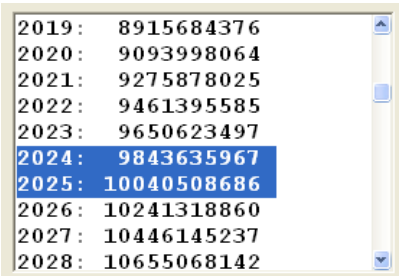
2) Brukeren kan taste inn noe som ikke er tall. Gjør om slik at det ikke gir kjørefeil, men evt. gir en feilmelding uten å beregne noe.

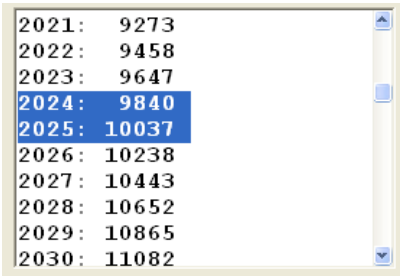
3) Legg til *ToolTips* for alle tekstkontrollene og kommandoknappen:



Vink: Du må legge til en *ToolTip* kontroll og fylle ut egenskapen *ToolTip* for hver kontroll.

Forsøk å få tallene i resultatet pent "høyrestilt" under hverandre, så det ser slik ut der antallet siffere øker:





Vink: Du må bruke en skrifttype (font) som har fast bredde på tegnene. En slik er f.eks. *Courier New*. Videre må du legge til et antall mellomrom foran befolkningstallet, avhengig av hvor mange siffere det er i tallet. Funksjonen *s.PadLeft(n)* der *s* er en String og *n* er et heltall, vil legge til et passende antall mellomrom forrest i *s* så den blir *n* lang. F.eks. vil *"Knut".PadLeft(8)* bli *"   Knut"* (4 blanke foran Knut).
Prøv å gjøre det så generelt som mulig, så det også virker uansett antall siffere! Da bør du finne ut hva befolkningen er det siste året (*sluttår*) og bruke antall tegn i det tallet til "paddingen". Befolkningen for siste år er gitt ved formelen

sluttbefolkning = CLng(befolkning * (1 + vekst) ^ (sluttår - startår))

Kapittel 7 – Vi bruker datakraften

Datamaskiner har en fenomenal regnekraft. Hvis vi har et problem som krever mye regning – eller mye prøving og feiling – må vi bare ta oss tid til å skrive ett program så kan maskinen på kort tid finne løsningen.

Eksempel 1: Hamar-mart'n (om haner, høner og kyllinger)

Jeg begynner med et eksempel der algoritmen er enkel. Eksempelet ble kort omtalt som øvelse i numerisk analyse i kapittel 5. Det gjelder han far som skal til Hamar med slakt, og vil benytte anledningen til å kjøpe fjærkre. Han vil ha akkurat 100 dyr og de skal koste akkurat 100 kroner. Han vil dessuten ha minst én av hvert slag.

På markedet finner han haner for kr 10 pr stykk, høner for kr 5 pr stykk og kyllinger for 50 øre pr stykk.

Hvor mange må han kjøpe av hvert slag?

Løsning 0 – "rett fram uten å tenke"

Hvis vi ikke tenker oss om, kan vi starte med 1 hane og 1 høne og 1 kylling og kontrollere om det blir 100 stykk og koste 100 kroner. Hvis ikke så prøver vi med 1 haner, 1 høne og 2 kyllinger osv. opp til 200 kyllinger. Da prøver vi 1 hane, 2 høner og 1 kylling. Slik holder vi på til en løsning er funnet.

Det blir i alt $10 \cdot 20 \cdot 200 = 40\,000$ muligheter og kan ta litt tid å gjøre manuelt☺! Med datamaskin går det raskt:

```
'Rett fram - brute force og ingen optimalisering
Dim haner, høner, kyllinger As Integer
For haner = 0 To 10 '10 er maks - koster 100 kr
   For høner = 0 To 20 '20 er maks - koster 100 kr
      For kyllinger = 0 To 200 '200 er maks - koster 100 kr
         If haner * 10 + høner * 5 + kyllinger * 0.5 = 100 _
            And haner + høner + kyllinger = 100 Then 'funnet løsning
            MsgBox("Du må kjøpe " & haner & " haner, " _
                  & høner & " høner og " & kyllinger & " kyllinger!")
         End If
      Next
   Next
Next
```

Løsning 1

Selv om dette gikk raskt, er det mulig å forbedre algoritmen. Hvis tallene hadde vært større, eller oppgaven skulle gjøres mange gang i sekundet, vil det lønne seg.

Vi forstår lett at de må bli [1..9] haner fordi 10 haner koster så mye at det ikke blir råd til høner og kyllinger også. Tilsvarende [1..19] høner og [1..199] kyllinger. Videre vet vi at det skal være maksimalt 100 fjærkre, altså har det ingen hensikt å fortsette når vi har nådd 98 kyllinger (+ 1 hane + 1 høne = 100).

Vi kan starte med 1 hane og 1 høne og 1 kylling og kontrollere om det blir 100 stk og koste 100 kroner. Hvis ikke så prøver vi med 1 haner, 1 høne og 2 kyllinger osv. til 98 kyllinger. Da prøver vi 1 hane, 2 høner og 1 kylling. Slik holder vi på til en løsning er funnet.

Det blir i alt $9 \cdot 19 \cdot 98 = 16.758$ muligheter, altså betydelig færre med bare *litt* tenking.

```
'Optimalisert noe
Dim haner, høner, kyllinger As Integer
For haner = 0 To 9 '10 er maks - koster 100 kr
   For høner = 0 To 19 '20 er maks - koster 100 kr
      For kyllinger = 0 To 98 '200 er maks - koster 100 kr
         If haner * 10 + høner * 5 + kyllinger * 0.5 = 100 _
            AndAlso haner + høner + kyllinger = 100 Then 'funnet løsning
            MsgBox("Du må kjøpe " & haner & " haner, og " _
                  & høner & " høner og " & kyllinger & " kyllinger!", , "Løsning")
         End If
      Next
   Next
Next
```

Legg merke til "kortslutningen" av det logiske uttrykket med *AndAlso* (jfr side

Løsning 2

Vi kan starte med 1 hane. Da er det bare $(100 - 10)$ kroner igjen å kjøpe høner for. Det blir maks $\frac{100-10}{5}$ høner, da er det ingen penger igjen. Generelt kan vi maksimalt kjøpe $\frac{100-10 \cdot haner}{5}$ høner. Når vi har kjøpt haner og høner er det enda mindre penger igjen, nemlig $(100 - 10 \cdot haner - 5 \cdot høner)$. Derfor kan vi kjøpe maks $\frac{100-10 \cdot haner-5 \cdot høner}{0.5}$ kyllinger.

Dette kan vi utnytte til å stanse mye tidligere i løkkene:

```
'Denne er optimalisert mer
Dim haner, høner, kyllinger As Integer
For haner = 0 To 9 '10 er maks - koster 100 kr
   For høner = 0 To (100 - haner * 10) / 5 'noe av pengene er brukt opp
      For kyllinger = 0 To (100 - haner * 10 - høner * 5) / 0.5 'enda mer er brukt
         If haner * 10 + høner * 5 + kyllinger * 0.5 = 100 _
            And haner + høner + kyllinger = 100 Then 'funnet løsning
            MsgBox("Du må kjøpe " & haner & " haner, og " _
                  & høner & " høner og " & kyllinger & " kyllinger!", , "Løsning")
         End If
      Next
   Next
Next
```

Løsning 3

Hvis vi regner litt først, blir det hele enda raskere:

$Formel\ 1: haner + høner + kyllinger = 100$

$\Rightarrow kyllinger = 100 - haner - høner$

$Formel\ 2: 10 \cdot haner + 5 \cdot høner + 0.5 \cdot kyllinger = 100$

$Formel\ 3\ (1\ innsatt\ i\ 2): 10 \cdot haner + 5 \cdot høner + 0.5(100 - haner - høner) = 100$

$$\Rightarrow høner = \frac{50 - 9.5 \cdot haner}{4.5}$$

Det blir da bare én løkke: Prøv med 1 hane, beregn antall høner etter formel 3 og deretter antall kyllinger etter formel 1 ovenfor, og sjekk så om en løsning er funnet. Hvis ikke så prøv med 2 haner osv til 9 haner. Det blir bare 9 iterasjoner!

Hvis vi vet at det bare blir én, mulig løsning, kan vi dessuten hoppe ut av iterasjonen når den er funnet. (Ikke si dette til Thor – han *hater* slike uthopp!)

```
'Denne er optimalisert så langt jeg klarer
Dim haner, høner, kyllinger As Integer
For haner = 1 To 9 'iterasjon
   høner = (50 - 9.5 * haner) / 4.5 'formel 2
   kyllinger = 100 - haner - høner 'formel 3
   If haner * 10 + høner * 5 + kyllinger * 0.5 = 100 _
      And haner + høner + kyllinger = 100 Then 'funnet løsning
      MsgBox("Du må kjøpe " & haner & " haner, og " _
            & høner & " høner og " & kyllinger & " kyllinger!", , "Løsning")
      Exit Sub ' funnet den ene løsningen
   End If
Next
```

☺ Kuult, esse!

Eksempel 2: Flate under graf

Det er ofte behov for å finne flateinnholdet mellom grafen for en funksjon og x-aksen – vi kaller det gjerne "arealet under grafen". Ofte kan det finnes analytisk (ved regning) ved å bruke integralregning ("antiderivert"), men det er vanskelig og ikke alltid mulig. Da trår vi IT-folk til igjen og bruker datakraften vår til å finne et tilnærmet svar med numerisk metode.

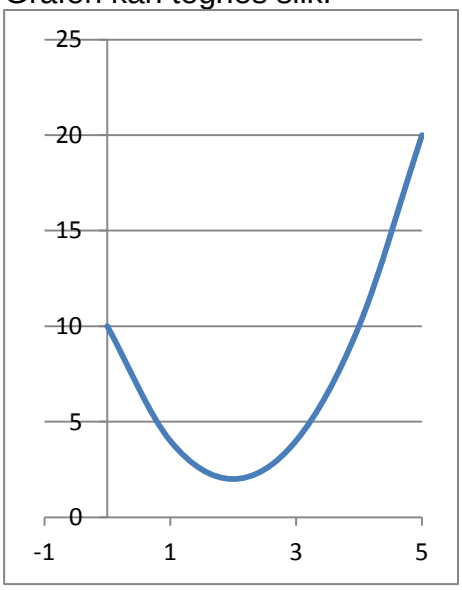
Eksempel

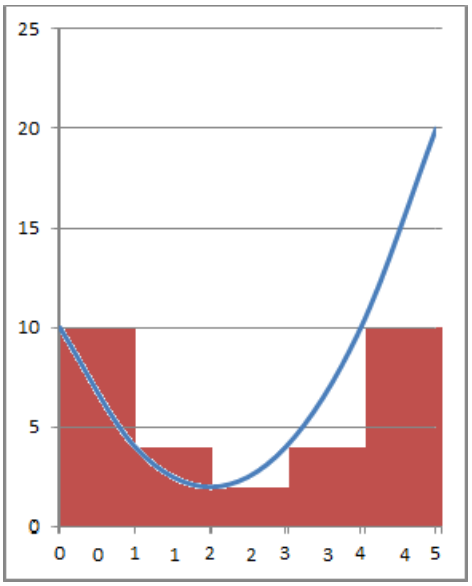
Vi har funksjonene $y = f(x) = 2x^2 - 8x + 10$. Vi er interessert i flateinnholdet under grafen fra x=0 til x=5. Integralregning – som her er mulig – gir svaret $33\frac{1}{3} \approx 33{,}333$. (Jeg viser ikke her hvordan det finnes.)

Vi tabulerer litt for å få tegnet den:

x	0	1	2	3	4	5
y	10	4	2	4	10	20

Grafen kan tegnes slik:





Til venstre er den tegnet kontinuerlig, til høyre har jeg tegnet inn søyler som tilsvarer grafens høyde dvs. f(x) for hver hele x. Hvis jeg nå summerer flateinnholdet av søylene som hver har bredde 1, får jeg:

$$F = f(0) + f(1) + .. + f(4) = 10 + 4 + 2 + 4 + 10 = 30$$

Dette er mitt foreløpige estimat for flateinnholdet F og det er litt lite.

Det er dessuten svært unøyaktig. Hvis jeg lager søylene smalere, vil tilpasningen bli bedre. Hvis hver av dem har bredde 0,2 får jeg f.eks.:

$$F = f(0) \cdot 0{,}2 + f(0{,}2) \cdot 0{,}2 + f(0{,}4) \cdot 0{,}2 + .. + f(4{,}8) \cdot 0{,}2$$
$$= 10 \cdot 0{,}2 + 8{,}48 \cdot 0{,}2 + \cdots + 17{,}68 \cdot 0{,}2 \approx 32{,}4$$

som er nærmere det riktige resultatet.

Med datamaskin kan vi enkelt lage søylene svært tynne og tilsvarende svært mange av dem og finne et ganske bra resultat. Slik ser hovedalgoritmen da ut:

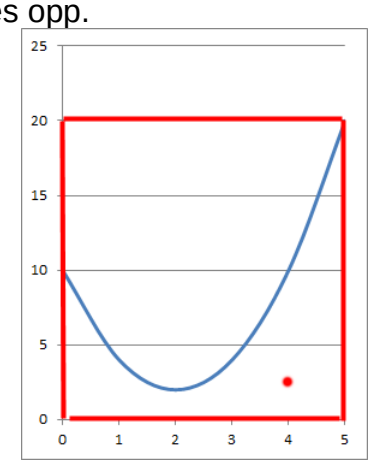
```
Dim Trinn As Double 'nøyaktighet
Dim Flate1 As Double
Dim X As Double
Dim Y As Double

'Initier variable:
Trinn = 5 / Val(txtAntallSøyler.Text) 'beregner søylebredden
Flate1 = 0

'Gå gjennom intervallet, trinn for trinn
For X = 0 To 5 - Trinn / 2 Step Trinn  'Trinn/2 pga regneunøyaktighet
   Y = 2 * X ^ 2 - 8 * X + 10 'beregn f(X)
   Flate1 = Flate1 + Trinn * Y 'Akkumuler Flate1 med den nye søylen
Next X
```

Med 10.000 søyler som hver er 0,0005 brede, blir svaret 33,33308 som er svært nær.

Løsning 2

En annen metode er å bruke tilfeldigheter. Vi tenker slik: Vi tegner grafen på et stort papir og fester det på veggen. Med ryggen til skyter vi så med hagle mot grafen. Noen av skuddene treffer under grafen, andre over. Vi teller opp og finner f.eks. at hver tredje hagel traff under grafen. Da utgjør flateinnholdet under grafen omtrent 1/3-del av hele arket på $5 \cdot 20$. Vi teller ikke de som treffer utenfor arket.

Vi simulerer dette ved å beregne x = rnd()*5 og y=rnd()*20. Hvis f(x)<=y, så traff hagelen under grafen og telles opp.



I det røde punktet er f.eks. (x,y)=(4,3). Siden f(4)=10 og y er trukket til 3, så traff vi her under grafen fordi y=3 er mindre enn y=f(4).

Hovedalgoritmen kan f.eks. være slik:

```
Dim Antall As Long  'antall som skal simuleres
Dim Treff As Long  'antall verdier under grafen
Dim SkuddNr As Long
Dim X As Double
Dim Y As Double
Dim Flate2 As Double
'Initier variable:
Antall = Long.Parse(txtAntall.Text)
Treff = 0
Randomize()
'Simuler tilfeldige "skudd"
For SkuddNr = 1 To Antall
   X = Rnd() * 5 'tilfeldig x-verdi 0-5
   Y = Rnd() * 20 'tilfeldig y-verid 0-20
   If Y <= 2 * X ^ 2 - 8 * X + 10 Then 'skuddet ligger under grafen
      Treff += 1 'tell opp "treff"
   End If
   Flate2 = 20 * 5 * Treff / SkuddNr 'beregn flateinnhold som andel av treff
Next SkuddNr
```

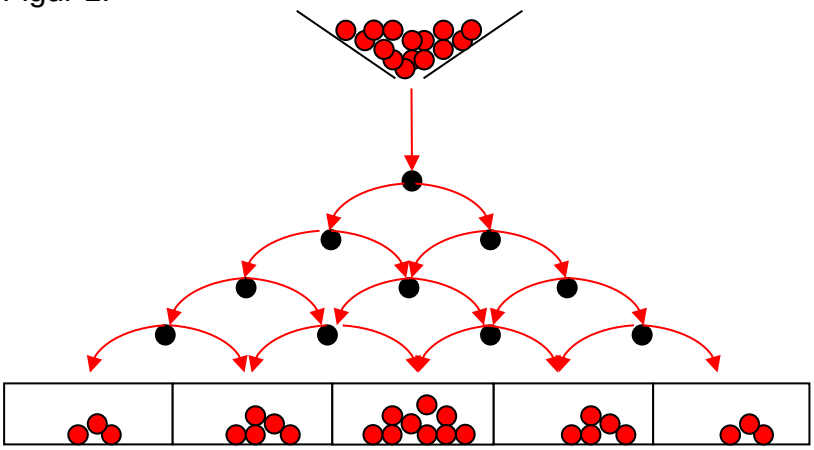
Når dette prøvekjøres vil svaret variere. Med 10.000 "skudd" fikk jeg følgende resultater:
32,92, 33,69, 33,17, 33,92 og 32,75. Gjennomsnittet av de fem svarene er 33,29. (Dette gjennomsnittet vil være normalfordelt så sjansen for å få et gjennomsnitt nær 33,33 er ganske stor.) Med større antall "skudd" i hver simulering vil resultatene variere mindre.

Eksempel 3: Fallende kuler

Røde kuler faller ned én og én og treffer stifter som er satt opp slik at kulen kan sprette like gjerne mot høyre som mot venstre. De ender nede i et antall skåler. Spørsmålet er da hvordan kulene vil fordele seg i skålene?

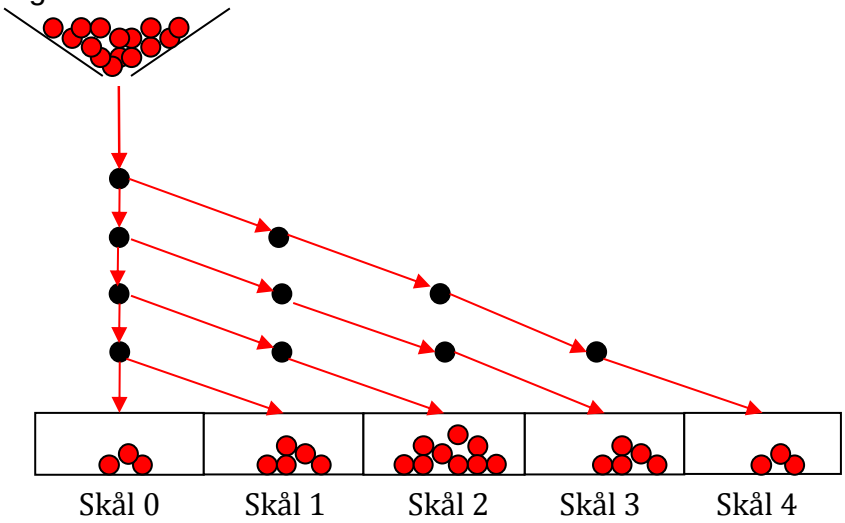
Dette kan statistikerne (og jeg☺) regne på analytisk, men det er mye artigere å simulere det på en datamaskin, så derfor lager vi et program. Da legger vi opp til at høyresprett har sannsynlighet ½, dvs når Rnd()>=0.5.

Figur 1:



Dette kan vi modellere litt annerledes. Da ser vi bare på sprett mot høyre ("høyresprett").

Figur 2:



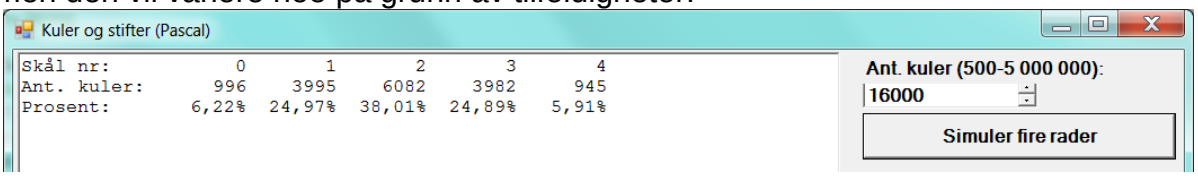
Man kan se at en kule som spretter mot høyre 0 ganger, vil havne i skålen helt til venstre – kalt skål 0. En kule som spretter 1 gang mot høyre, havner i skål 1 osv. opp til antall rader (4 rader med stifter gir fem skåler, nummerert 0, 1, ... 4.

Program

Kjernen i simuleringen ser slik ut:

```
'Programmert "rett fram" for fire rader
Dim Skål(4) As Integer
Dim Kule As Integer
Dim Sprett As Integer
Dim AntKuler As Integer = CInt(numAnt.Value)
Dim AntHøyre As Integer

'Simuler kulene og tell opp for hver skål
For Kule = 1 To AntKuler
   AntHøyre = 0
   For Sprett = 1 To 4 'fire rader med stifter
      AntHøyre = AntHøyre + Int(Rnd() * 2) '50/50 for 0 eller 1
   Next Sprett
   Skål(AntHøyre) += 1 'enda en kule har falt ned i denne skålen
Next Kule
```

Det kan passe å prøvekjøre med 16 000 kuler, da blir fordelingen ganske tydelig, men den vil variere noe på grunn av tilfeldigheter:



Teoretisk resultat er gjengitt i neste avsnitt og er 1000, 4000, 6000, 4000 og 1000.

Ekstra for spesielt interesserte: Analytisk løsning

Hvert sprett har to utfall og er uavhengig av foregående og sannsynligheten for høyresprett er konstant $= \frac{1}{2}$. Dette er da en binomisk fordeling $S \sim bin(n,p) = \sim bin(4,\frac{1}{2})$. Da er forventningen

$$E(S) = n \cdot p = 4 \cdot \tfrac{1}{2} = 2.$$

Vi kan forvente at kulene "gjennomsnittlig" havner i skål 2.

Sannsynligheten for at kulen skal falle i skål *s* er gitt ved:

$$P(S = s) = \binom{4}{s} \cdot p^s \cdot (1-p)^{n-s} = \binom{4}{s} \cdot \frac{1}{2}^S \cdot (1-\frac{1}{2})^{4-s} = \binom{4}{s} \cdot \frac{1}{16}$$

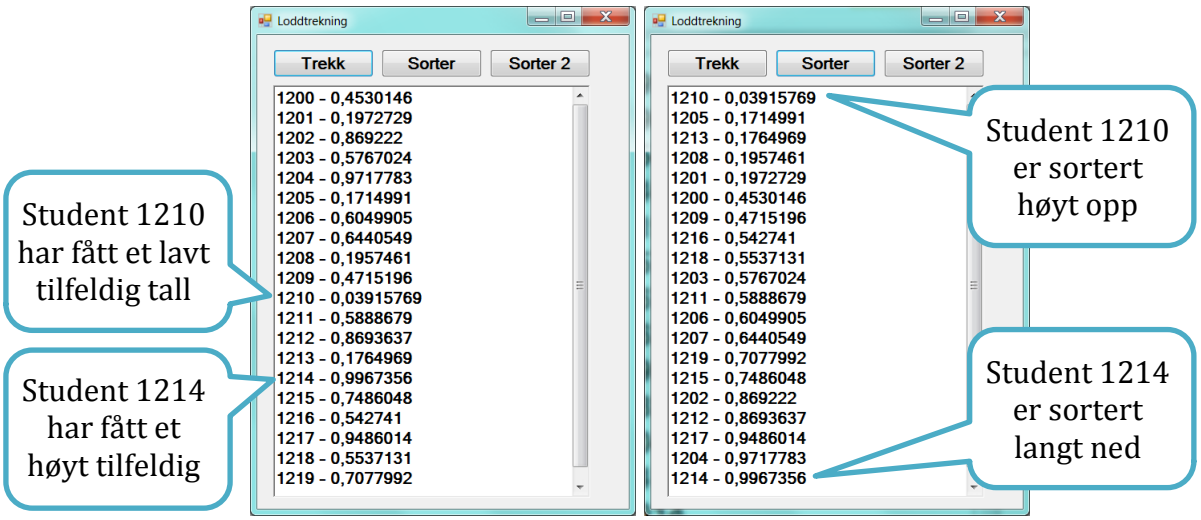
Da kan vi beregne hvordan kulene i det lange løp vil fordele seg i skålene, utfra sannsynlighetene:

$$P(S = 0) = \binom{4}{0}\frac{1}{16} = \left(\frac{4!}{0!4!}\right)\frac{1}{16} = \frac{1}{16} = 6{,}25\%$$
$$P(S = 1) = \binom{4}{1}\frac{1}{16} = \left(\frac{4!}{1!3!}\right)\frac{1}{16} = \frac{4}{16} = 25\%$$
$$P(S = 2) = \frac{6}{16} = 37{,}5\%$$
$$P(S = 3) = \frac{4}{16} = 25\%$$
$$P(S = 4) = \frac{1}{16} = 6{,}25\%$$

Du ser at resultatet av prøvekjøringen er ganske nær disse "riktige" verdiene.

Eksempel 4: Loddtrekning med array

Vi skal sortere en array med et antall studentnumre. Arrayen er fylt slik at studentnumrene står sortert etter størrelse, men den skal sorteres i tilfeldig rekkefølge.



Til venstre: Sortert etter nummer og tilordnet tilfeldige tall
Til høyre: Sortert etter de tilfeldige tallene

Ideen her er jeg først lager en array med et antall studentnr og til hvert studentnr knytter jeg et tilfeldig tall i intervallet [0..1>. Etterpå sorterer jeg etter det tilfeldige tall. Arrayen er deklarert slik:

```
Private studarray(1, 19) As Single
```

Den ser slik ut etter at den er fyllt:

studarray

	kolonne 0 (studentnr)	**kolonne 1 (rnd)**
rad 0	1200	0.3456
rad 1	1201	0.123
…	…	…
rad 19	1219	0.8865

```
Public Class Loddtrekning
    Const antstud As Integer = 20
    Private studarray(1, antstud - 1) As Single
    Private i, indeks As Integer

    Private Sub Loddtrekning_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
        'Fyll array med data
        Dim studnr As Single = 1200
        For indeks = 0 To antstud - 1
            studarray(0, indeks) = studnr 'studnr i kolonne 0
            studnr += 1
        Next
    End Sub

    Private Sub cmdTrekk_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles cmdTrekk.Click
        Randomize()
        'Trekk tilfeldig tall til hver student:
        For indeks = 0 To antstud - 1
            studarray(1, indeks) = Rnd() 'tilfeldig tall i kolonne 1
        Next
        visdata()
    End Sub

    Private Sub cmdSort_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles cmdSort.Click
        'Sorterer den doble arrayen
        For i = 0 To antstud
            For indeks = 0 To antstud - 2
                If studarray(1, indeks) > _
                    studarray(1, indeks + 1) Then 'skal byttes
                    byttom(studarray(0, indeks), studarray(0, indeks + 1))
                    byttom(studarray(1, indeks), studarray(1, indeks + 1))
                End If
            Next indeks
        Next i
        visdata()
    End Sub

    Private Sub byttom(ByRef a As Single, ByRef b As Single)
        Dim temp As Single
        temp = a
        a = b
        b = temp
    End Sub

    Private Sub visdata()
        'Vis dataene i tekstboksen
        rtbStud.Clear()
        For indeks = 0 To antstud - 1
            rtbStud.Text &= studarray(0, indeks).ToString() & " - " _
                & studarray(1, indeks).ToString() & vbCrLf
        Next
    End Sub
End Class
```

Eksempel 5: Sortere en array i tilfeldig rekkefølge ("stokke" elementene)

Ovenfor brukte jeg algoritmen "Boblesort" til å sortere arrayen. Den er ikke særlig effektiv. På http://www.sorting-algorithms.com/ kan du se en flott animering av de

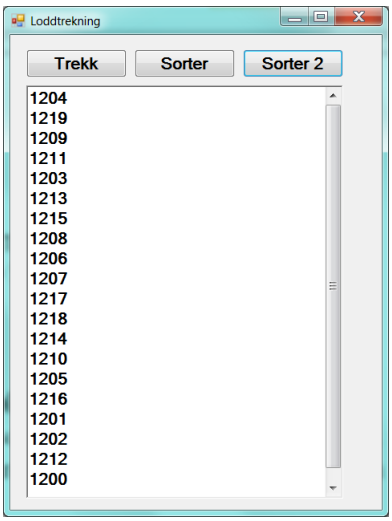
forskjellige sorteringsalgoritmene og hvor effektive de er under forskjellige forhold. Anbefales!

Det finnes flere andre og bedre algoritmer for tilfeldig sortering[11]. Dere kan f.eks. bruke Durstenfelds algoritme (kalt "Algorithm P" av Donald Knuth – sjekk om du forstår hva som foregår):

To shuffle an array *a* of *n* elements (indices 0..n-1):
for *i* **from** *n* − 1 **downto** 1 **do**
j ← random integer with 0 ≤ *j* ≤ *i*
exchange *a*[*j*] and *a*[*i*]

Vi skal sortere en array *stud(antStud-1)*. Koden kan bli slik:

```
Dim stud(antstud - 1) As Integer
Dim i, j, tmp As Integer
'Fyll arrayen med studentnumre fra 1200 og oppover
For i = 0 To antstud - 1
   stud(i) = 1200 + i
Next
'Sorter tilfeldig (Algorithm P)
For i = antstud - 1 To 1 Step -1 'teller nedover
   j = CInt(Rnd() * i)
   'Bytt to elementer
   tmp = stud(j)
   stud(j) = stud(i)
   stud(i) = tmp
Next
'Visdata (duplisert kode - ikke bra!
rtbStud.Clear()
For i = 0 To antstud - 1
   rtbStud.Text &= stud(i).ToString() & vbCrLf
Next
```



[11] Se f.eks. http://en.wikipedia.org/wiki/Fisher-Yates_shuffle

Ekstra: Variablenes synlighet

I dette programmet er det et hovedskjema *frmHoved*, et ekstra skjema *frmRegistrer* og en modul *modAntall*. En *modul* er en programdel uten synlig skjema, men oppfører seg ellers likt med et skjema.

```
Public Class frmHoved
    Dim studenter as ...

    Private Sub listAlle( )
        Dim utstreng as ...
        For i As ...
            <gjør noe>
        Next i
    End Sub

    Public Function finn( ) As ...
        Dim student As ...
        <gjør noe>
    End Function

End Class
```

```
Module modAntall
    Private antall As ...

    Public Sub økAntall ( )
        <gjør noe>
    End Sub

End Module
```

```
Public Class frmRegistrer
    Dim student As ...

    Private Sub lesData ( )
        <gjør noe>
    End Sub

End Class
```

- ✓ Det som er deklarert *Public* er synlig overalt i hele programmet.
- ✓ Det som er deklarert *Private* eller *Dim* er synlig bare der det er deklarert. F.eks. er variabelen studenter bare synlig for *listAlle()* og *finn()*. For-løkkens variabel *i* er bare synlig inne i for-løkken osv.
- ✓ Alle kontroller som tekstfelt. knapper, etiketter osv. på et skjema er synlige overalt i programmet – og eksisterer – så lenge skjemaet er lastet i RAM. (Dette går det an å gjøre noe med, men det er utenfor pensum.)
- ✓ Hvis du vil referere til noe synlig utenfor eget skjema, må du også angi skjemanavnet med punktnotasjon. F.eks. kan kode i prosedyren *lesData()* kalle på *finn()* ved å skrive *frmHoved.finn()*. Inne i *finn()* kan du *ikke* skrive *frmRegistrer.lesData()* eller *frmRegistrer.student* fordi de ikke er synlige utenfor skjemaet *frmRegistrer*. I begge skjemaene kan du skrive *modAntall.antall*.

Av hensyn til vedlikehold og for å kunne dele opp arbeidet med programmering mellom flere programmerere, vil vi gjerne ha minst mulig synlighet. Følg disse reglene så blir synligheten minst mulig:

Regel 1: Deklarer variable ”på lavest mulig nivå”
Regel 2: Bruk *Dim* og *Private* mest mulig

Hvis flere elementer heter det samme, brukes den ”nærmeste”, hvis du ikke angir noe annet med punktnotasjon. Eksempel:

```
Public Class frmKarakterer 'skjema
   Dim i As Integer
   Private Sub X()
      Dim i As Integer
      i = 15 'den lokale i
      Me.i = 20 'skjemavariabelen i (kan ikke skrive frmKarakter her)
      frmStat.i = 30 'en public variabel på skjema frmStat
      frmStat.txtHilsen.Text = "hei" 'en kontroll på skjema frmStat
   End Sub
End Class
```

Selv om *Dim* er lovlig og det samme som *Private*, bør du helst skrive *Public* eller *Private* på skjema- og modulnivå. Inne i en prosedyre og funksjon får du bare lov til å skrive *Dim*. Det har jo ingen hensikt å skrive *Public* siden variabelen eksisterer i RAM bare mens prosedyren/funksjonen kjører og da kan ingen andre bruke den til noe (programmet er opptatt med å kjøre denne prosedyren/funksjonen).

Dim er kun lovlig for variable. For funksjoner/prosedyrer må du bruke *Private*/*Public* det er ikke lovlig med *Dim. H*vis du ikke skriver noe, blir de automatisk *Public* men du bør venne deg til å tenke deg om og så skrive *Private* eller *Public*.

Søk i VB-hjelp etter ”Dim” og ”Lifetime in Visual Basic” for å få alle, detaljerte regler.

Oppgave til kapittel 7 (rekkesum)

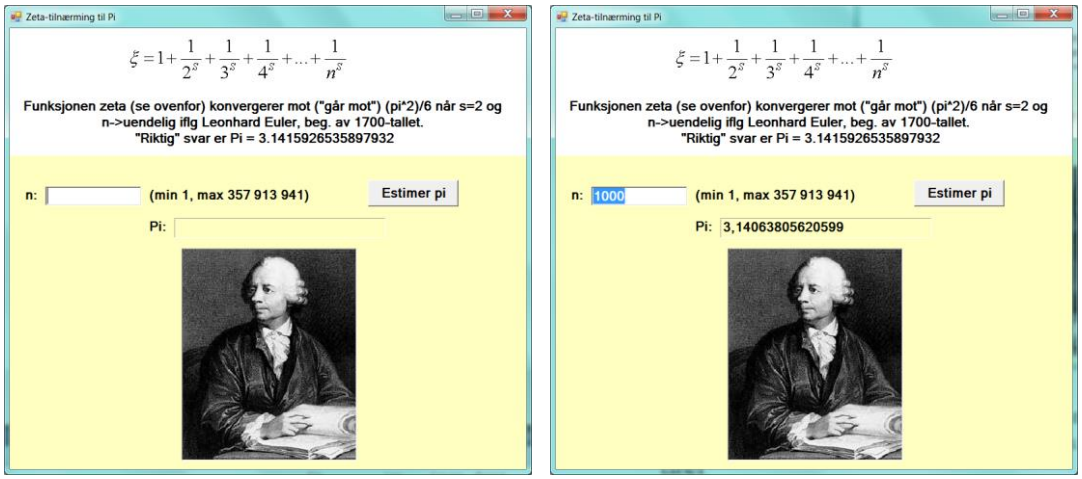
I begynnelsen av 1700-tallet viste professor Leonhard Euler at med en bestemt rekke, kunne man tilnærmet finne verdien av π (pi).

Rekkens ledd skal summeres, og summen ser slik ut (tegnet ξ er det greske tegnet *zeta*) og har *n* ledd:

$$\xi = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots + \frac{1}{n^2} = \frac{1}{1} + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \frac{1}{25} + \cdots$$

Summen ξ nærmer seg ikke π direkte når *n* økes, men nærmer seg $\frac{\pi^2}{6}$. Når du har funnet summen, må du altså første gange summen med 6 og deretter ta kvadratroten av svaret.

For å få en noenlunde bra resultat, må antallet ledd, *n*, være stort. Med datatidens fjærpenner og manuelle regnearbeid, var det en stor jobb å komme noen særlig vei med dette. I dag kan vi bruke datamaskinen til å finne flere hundre millioner ledd i rekken og innebygd funksjon til å finne kvadratroten. Det er det du skal gjøre nå.



I et tekstfelt lar du brukeren bestemme hvor mange ledd, *n*, som skal tas med. Kontroller at *n* er minst 1 og maks 357 913 941. Summen må være deklarert som en double. Svaret viser du i en label. Prøvekjør med forskjellige antall ledd.

Vedlegg: Prøveeksamen

Nedenstående prøveeksamen er tilpasset kurset INF150 Grunnleggende programmering ved Høgskolen i Buskerud og er årlig gitt der. Nedenstående oppgave gjelder bare begrepsdelen. På eksamen vil en stor del av oppgavesettet gjelde programmering så det er mindre om begreper.

INF150 Grunnleggende programmering - Begreper
BOKMÅLSTEKST

Eksamenstid: 4 timer
Hjelpemidler: Ingen
Oppgaveteksten består av 3 sider. Alle oppgaver skal besvares.
Begynn gjerne besvarelsen av hver oppgave på nytt ark.
Note: I alle beregningsoppgaver og liknende, skal du vise hvordan du finner svarene – riktig svar godtas ikke alene.

Oppgave 1: Mengder

Spørsmål A

For dem som er vant til en annen notasjon, opplyses det om at symbolet ¬ foran en mengde tilsvarer en strek over den, f.eks. $\neg x = \bar{x}$.

A, B og C er navnet på tre mengder som delvis overlapper hverandre.

a) Hva er en mengde?
b) Forklar med ord og tegning (Venndiagram) hva vi mener med uttrykket $A \cup (B \cap C)$.
c) Vis med Venndiagrammer at uttrykket $(\neg A \cup \neg B \cup \neg C)$ er det samme som uttrykket $\neg(A \cap B \cap C)$.
d) Hvis $A \cap B = B$, hva er da $A \cup B$ og $\neg A \cap B$? [**Vink:** Lag et Venndiagram som tilfredsstiller det første uttrykket, så ser du nok svaret på de to andre direkte.]

Spørsmål B

Vi har to mengder M={3, 2, 5} og F={9, 2}. Hva er da

a) $M \cup F$?
b) $M \cap F$?
c) $M \times F$ og hva uttrykker det?

Oppgave 2 - Logikk

For dem som er vant til en annen notasjon, opplyses det om at symbolet ¬ foran en mengde tilsvarer en strek over den, f.eks. $\neg A = \bar{A}$.

x, y og z er logiske (Boolske) utsagn som er uavhengige av hverandre.

Spørsmål A

Vis med sannhetsverditabell at uttrykket $[(y = True) \wedge (z = True)]$ er det samme som uttrykket $(y \wedge z)$.

Spørsmål B
Forklar kort hvordan vi kan utnytte resultatet i spørsmål (a) ved programmering av seleksjoner (valg, altså if-setninger).

Spørsmål C
Sett opp sannhetsverditabell for uttrykket $(\neg x \wedge y) \vee \neg z$

Oppgave 3 – Tallsystemer
Spørsmål A
Vis hvordan det heksadesimale tallet $E9_H$ kan omgjøres til både binært (grunntall 2) og desimalt (grunntall 10) tall.

Spørsmål B
Vis hvordan det desimale tallet 29_{10} skrives binært og heksadesimalt.

Spørsmål C
a) Sett opp regnestykket og regn ut differansen $11000010_2 - 10101_2$.
b) Sett opp regnestykket og regn ut divisjonen $110011_2 : 1011_2$.

Tallene skal *ikke* gjøres om til andre tallsystemer.

Spørsmål D
Kan man ha et tallsystem med grunntall 1? Forklar svaret. [**Vink:** Hvor mange og hvilke siffer må et slikt tallsystem ha?]

Spørsmål E
I en 16 bits maskin brukes toerkomplement for negative tall. Hvordan blir koden for tallet -9_D i en slik maskin?

Oppgave 4 – Funksjoner
Spørsmål A
Det skal lages et program som simulerer mange kast med to terninger. I den forbindelse er det laget denne "funksjonsstubben":

```
Private Function erLike() As Boolean
   '1. Simulerer kast av to terninger.
   Dim terning1 As Integer 'må få verdi!
   Dim terning2 As Integer 'må få verdi!
   '2. Returner True hvis terningene viser likt, ellers False
   Return True 'må endres!
End Function
```

Gjør ferdig funksjonen så den virker som forutsatt.

Spørsmål B
I Visual Basic er funksjonen *Rnd()* en *Single* som returnerer en "tilfeldig" verdi i intervallet
[0..1>.

a) Hva er et *intervall* i denne forbindelse?
b) Hva er forskjellen på det angitte intervallet for *Rnd()* og intervallet [0..1]?

Spørsmål C
I funksjoner anvendes variabler og konstanter.

a) Hva er en funksjon (definer)?
b) Hva er en variabel og en navngitt konstant (definer)?
c) Hva vil det si at en variabel *determinerer* enn annen? Forklar kort, gjerne med en figur.

Spørsmål D
Gitt funksjonen y = f(x) = 3лx - 6л (tre pi gange x, minus seks pi) som er en rett linje.

Hvor skjærer funksjonen f(x) x-aksen og y-aksen? Regn ut og forklar.

Spørsmål E
Vi ønsker å finne flateinnholdet under grafen til funksjonen g(x)=sin(x) i intervallet [0..π] med et program.

En måte å finne dette flateinnholdet på, er å tenke seg et antall ”søyler” under grafen og summere flateinnholdet av disse søylene. Ved å lage stadig smalere søyler, vil denne summen bli mer og mer lik flateinnholdet under grafen.

a) Tegn figur. [**Vink:** *sin(0)=sin(π)=0* og *sin(½π)=1*.]
b) Skriv programmet – bruk gjerne 0,0001 som søylebredde.
c) Vil ditt program gi teoretisk helt nøyaktig svar? Begrunn svaret.

Slutt på oppgavesettet

Tillegg: Siden dette er en *prøveeksamen*, anbefaler jeg at du *etter* å ha løst oppgave 4A og 4E på papir, skriver programmene inn på maskin og prøvekjører. Du bør få at omtrent 1/6-del er like i oppgave 4A, og flateinnholdet skal bli ca. 2 i oppgave 4E.