NTNU

Kunnskap for en bedre verden

TDT4105 Informasjonsteknologi, grunnkurs

Matlab 5: Løkker (FOR og WHILE)

Amanuensis Terje Rydland
Kontor: ITV-021 i IT-bygget vest (Gløshaugen)
Epost: terjery@idi.ntnu.no
Tlf: 735 91845

Bruk piazza for å få rask hjelp til alles nytte!

Ikke registrert deg? Gå inn på piazza.com og trykk på 'Sign up'

Allerede registrert deg? Gå inn på piazza.com og still spørsmål!

Læringsmål

- Løkker
  - FOR-setningen
  - WHILE-setningen
- Kapittel 5 i Matlab-boka

- Neste time: Algoritmeformulering

FOR-løkker

```
FOR <tellevariabel> = <start>:<inkrement>:<slutt>
    <setninger>
END
```

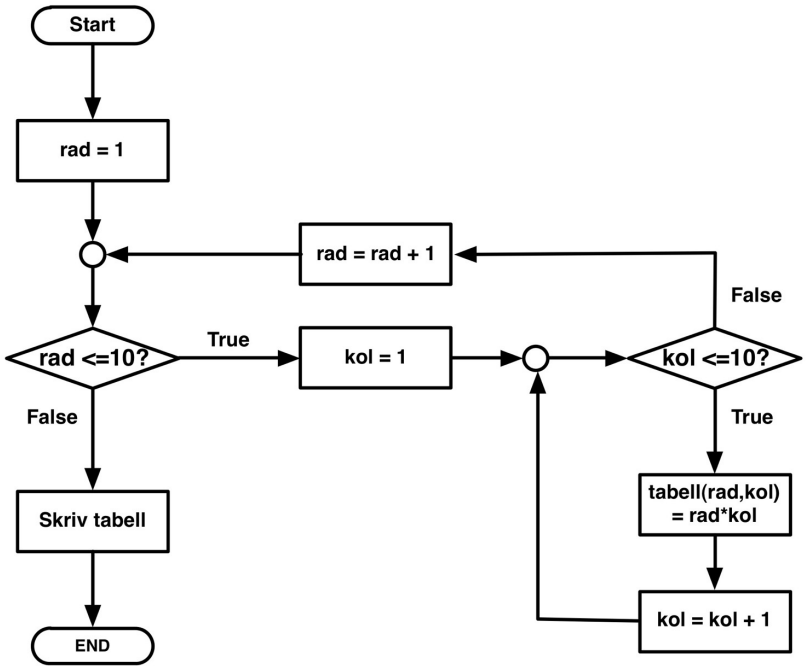
- Setningene i løkke-kroppen gjentas en gang for hver verdi av tellevariabelen

```
FOR <tellevariabel> = <tabell>
    <setninger>
END
```

- Løkkekroppen utføres en gang for hvert element i tabell

5

Flytdiagram - gangetabell



NTNU

6

gangetabell.m

```
% tømmer kommandovinduet
clc

for rad=1:1:10
    for kol=1:1:10
        tabell(rad, kol) = rad*kol;
    end
end

disp(tabell);
```

```
     1     2     3     4     5     6     7     8     9    10
     2     4     6     8    10    12    14    16    18    20
     3     6     9    12    15    18    21    24    27    30
     4     8    12    16    20    24    28    32    36    40
     5    10    15    20    25    30    35    40    45    50
     6    12    18    24    30    36    42    48    54    60
     7    14    21    28    35    42    49    56    63    70
     8    16    24    32    40    48    56    64    72    80
     9    18    27    36    45    54    63    72    81    90
    10    20    30    40    50    60    70    80    90   100
>>
```

NTNU

7

Telle forekomster av verdi i tabell

- Finne størrelsen på tabellen (dimensjonene)
  - size(<tabell>) returnerer antall rader og kolonner i <tabell>.
  - [rader, kolonner] = size(<tabell>)
- Ingen forekomster i utgangspunktet (antall = 0)
- Gå så gjennom tabellen
  - Rad for rad, kolonne for kolonne
  - Øke antall med en for hvert "treff"
- Har resultatet (i antall) når alle elementene er sjekket

2	1	3	2
4	1	1	3
1	4	4	2
2	3	3	1

NTNU

8

```
function antall = antallAvVerdi(tabell, verdi)
% teller antall forekomster av verdi i tabell

   % sjekker størrelsen på tabellen
   [antRader, antKolonner] = size(tabell);


   % startverdi
   antall = 0;

   % går gjennom tabellen
   for rad = 1:antRader
      for kol = 1:antKolonner

         if tabell(rad,kol) == verdi
             antall = antall + 1;
         end

      end
   end
end % function
```

NTNU

```
>> m = randi(3,10,10)
m =
     3     2     1     3     3     2     1     1     2     1
     1     2     3     1     1     3     1     3     1     3
     1     3     2     2     3     2     2     1     1     1
     1     3     1     3     1     2     3     2     3     3
     1     1     1     3     3     3     3     1     1     3
     3     2     2     3     2     1     1     2     3     3
     3     2     3     2     1     3     2     1     2     1
     1     2     2     1     1     3     2     2     3     2
     3     3     2     1     2     2     1     3     1     1
     1     3     1     1     2     2     2     3     2     3
>> antallAvVerdi(m,8)
ans =
     0
>> antallAvVerdi(m,1)
ans =
    36
>> antallAvVerdi(m,2)
ans =
    30
>> antallAvVerdi(m,3)
ans =
    34
>>
```


Summerer de 10 minste primtallene

primtallSum.m

```
% Skriptet summerer de 10 minste primtallene

% Tømmer kommandovinduet, fjerner alle variabler
clc, clear


primtall = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29];

sum = 0;

for i = primtall
   sum = sum + i;
   fprintf('Har lagt til %d, summen er nå %d\n', i, sum)
   pause(1) % venter i 1 sekund
end

fprintf('\nSluttsummen ble %d\n', sum)
```

Programkjøring

```
Har lagt til 2, summen er nå 2
Har lagt til 3, summen er nå 5
Har lagt til 5, summen er nå 10
Har lagt til 7, summen er nå 17
Har lagt til 11, summen er nå 28
Har lagt til 13, summen er nå 41
Har lagt til 17, summen er nå 58
Har lagt til 19, summen er nå 77
Har lagt til 23, summen er nå 100
Har lagt til 29, summen er nå 129
```


WHILE-løkken

```
WHILE <logisk betingelse>
    <setninger>
END
```

- Løkke-kroppen gjentas 0-N ganger
  - Hvis og så lenge betingelsen er sann
  - Løkkekroppen må ***påvirke*** betingelsen (etter hvert)
  - Antall ***iterasjoner*** er ofte ukjent på forhånd
- WHILE er mer generell enn FOR-løkken

Gjett et hemmelig tall

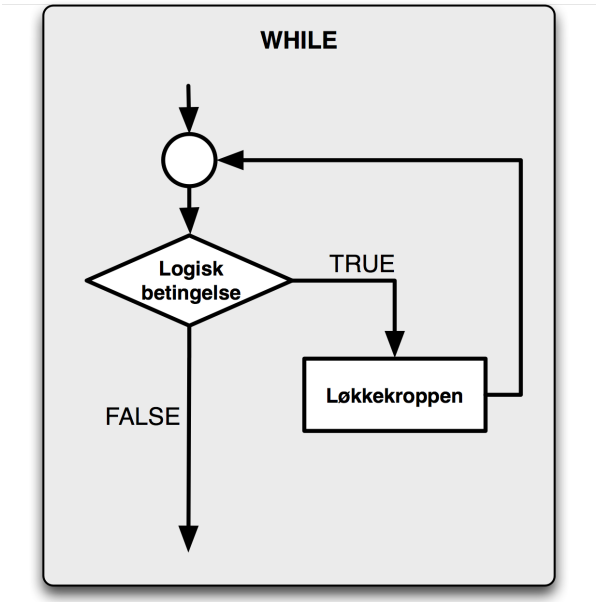
gjett_tall.m

```
% gjett et tall

maxTall = 10; % det stoerste tallet

tall = randi(maxTall); % trekker tallet

gjetter = true;
antallForsok = 0;

while gjetter

    forsok = input(['Gjett et tall fra 1 til ' num2str(maxTall) ': ']);
    if forsok == tall
        gjetter = false;
    else
        disp('Feil, prøv igjen');
    end
    antallForsok = antallForsok + 1;
end % while

fprintf('Riktig! Du klarte det på %d forsøk\n', antallForsok)
```



```
>> gjett_tall
Gjett et tall fra 1 til 10: 1
Feil, prøv igjen
Gjett et tall fra 1 til 10: 2
Feil, prøv igjen
Gjett et tall fra 1 til 10: 3
Feil, prøv igjen
Gjett et tall fra 1 til 10: 4
Feil, prøv igjen
Gjett et tall fra 1 til 10: 5
Feil, prøv igjen
Gjett et tall fra 1 til 10: 6
Feil, prøv igjen
Gjett et tall fra 1 til 10: 7
Riktig! Du klarte det på 7 forsøk
```

WHILE flytdiagram



Beregne pi

- Leibniz' formel

- Går i løkke, legger til et nytt ledd for hver iterasjon
- Når kan vi stoppe?
  - Når vi har lagt til et (fast) antall ledd.
  - Når det blir liten nok forskjell på to etterfølgende estimater.

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

piFunksjon

```
function nyPi = piFunksjon(epsilon)
% beregner pi med Leibnitz' formel
% avslutter når forskjellen på to påfølgende estimat
% er mindre enn epsilon


    forrigePi = 4;

    % k1
    k = 1;
    fortegn = -1;
    nyPi = forrigePi + fortegn*4/(2*k+1);

    while abs(nyPi - forrigePi) > epsilon
        forrigePi = nyPi;
        k = k + 1;
        fortegn = -fortegn;
        nyPi = forrigePi + fortegn*4/(2*k+1);
    end % while

end % function
```


Test-skript: testPiFunksjonen.m

```
clear, clc

epsilon = 1;

fprintf('epsilon        beregnet pi    avvik           tid\n')
for i=1:9
    epsilon = epsilon/10;
    tic
    beregnetPi = piFunksjon(epsilon);
    tid = toc;
    fprintf('%12.10f:  %12.10f %14.10f %10.5f\n', ...
        epsilon, beregnetPi, beregnetPi-pi, tid)
end
```

Testkjøring

```
epsilon         beregnet pi     avvik            tid
0.1000000000:   3.1891847823    0.0475921287     0.00004
0.0100000000:   3.1465677472    0.0049750936     0.00001
0.0010000000:   3.1420924037    0.0004997501     0.00006
0.0001000000:   3.1416426511    0.0000499975     0.00056
0.0000100000:   3.1415976536    0.0000050000     0.00466
0.0000010000:   3.1415931536    0.0000005000     0.04156
0.0000001000:   3.1415927036    0.0000000500     0.39661
0.0000000100:   3.1415926486   -0.0000000050     3.96106
0.0000000010:   3.1415926531   -0.0000000005    39.77501
>>
```


Tidtaking i Matlab

- tic – starter klokka
- toc – stopper klokka
- Se help tic / toc for mer informasjon
- NB! Ikke veldig nøyaktig.
  - Datamaskinen kan bruke tid på noe annet.

21

Evige løkker

- Løkker som *aldri* terminerer (avslutter)
  - `while true`
  - `while <noe som aldri blir usant>`
- ctrl-C (trykk ned både ctrl og C, samtidig)
  - Avslutter det som kjører i Matlab
- Ting å passe på:
  - Oppsett av "tilstand" før løkka
  - Endring av "tilstand" i løkkekroppen
  - MÅ ha et resonnement for terminering etter hvert

NTNU

22

NTNU

Kunnskap for en bedre verden

Kunnskap for en bedre verde

TDT4105 Informasjonsteknologi, grunnkurs

Matlab 6: Problemløsning / Algoritmer

Amanuensis Terje Rydland
Kontor: ITV-021 i IT-bygget vest (Gløshaugen)
Epost: terjery@idi.ntnu.no
Tlf: 735 91845

Læringsmål og pensum

- Læringsmål
  - Problemløsning: Fra problem til kjørende program
  - Algoritmebegrepet
  - Algoritmeformulering
    - Flytskjema
    - Pseudokode
- Pensum
  - Foilene, litt i algoritmedelen av teoriboka (kommer tilbake til det i uke 44 og 45.

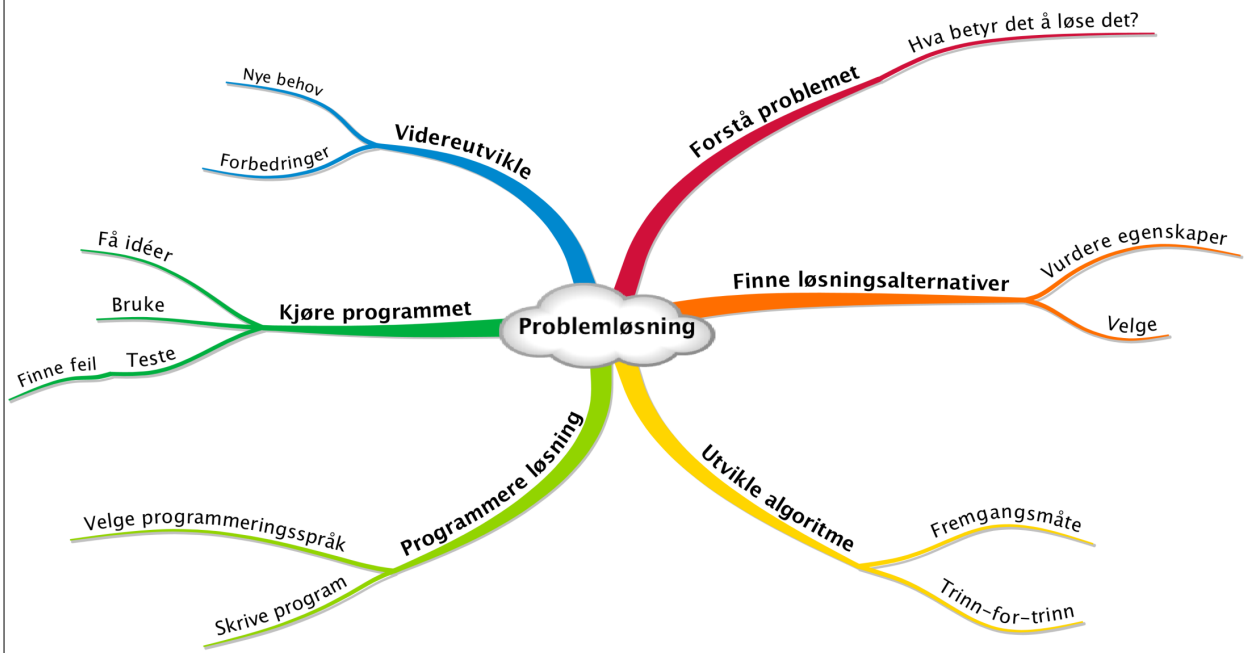
Algoritme

- I matematikk og informatikk er en algoritme en presis beskrivelse av en endelig serie operasjoner som skal utføres for å løse et problem. (Wikipedia)
- **Algoritme**, i matematikk og databehandling en fullstendig og nøyaktig beskrivelse av fremgangsmåten for løsning av en beregnings- eller annen oppgave. (Store norske leksikon)
- Omdannelse av navnet på den arabiske matematikeren **al-Khwarizmî** (ca. 820) i tilslutning til gresk **arithmós** (tall).

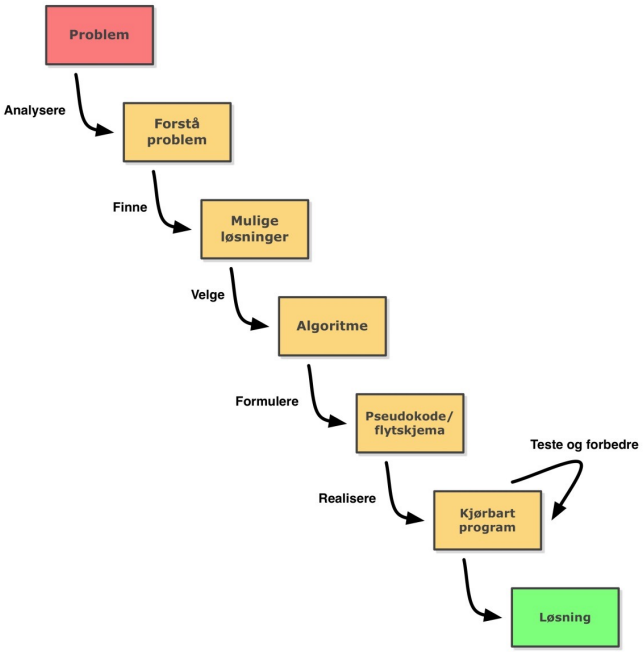
Euklids algoritme

- Euklids algoritme er en av de eldste kjente algoritmene, fra antikkens Hellas.
- Euklids algoritme for største felles faktor for to heltall (største heltall som deler begge) – sff(a,b)
  - Hvis et av tallene er 0 har vi løsning: sff(a,0) = a
  - Ellers trekker vi det minste tallet fra det største tallet helt til ett av tallene er lik null, det andre tallet er da svaret.
  - sff(35,21)=sff(14,21)=sff(14,7)=sff(7,7)=sff(7,0)
  - sff(9,11)=sff(9,2)=sff(7,2)=…=sff(1,2)=sff(1,1)=sff(1,0)
- NB! Finnes mer effektive algoritmer

Problemløsningsprosess



Problemløsningsprosess



Pseudokode

- Notasjon for å beskrive algoritmer som ligger mellom programmeringsspråk og naturlig språk.
- Benytter kontrollstrukturer fra programmeringsspråk for å formulere valg og repetisjon.
- Fordeler:
  - Tekst (kan skrives i editor)
  - Ligner strukturelt på løsning i programmeringsspråk
  - Raskt å skrive
  - Lett å endre
- Ulemper:
  - For likt programmer, stimulerer ikke andre deler av hjernen

29

sff i pseudokode

Les inn a og b

gjenta til a eller b er lik 0
    hvis a < b
        b = b – a
    ellers
        a = a – b
    slutthvis
sluttgjenta

svar = a + b

NTNU

30

sff i Matlab-kode (sff.m)

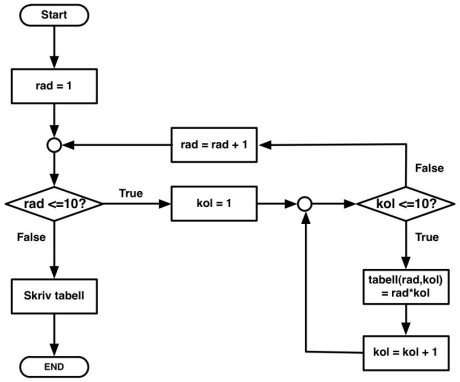
```
function f = sff(a, b)
% storste felles faktor for a og b

while (a ~= 0) && (b ~= 0)
    if a < b
        b = b - a;
    else
        a = a - b;
    end % if
end % while

f = a + b;

end % function
```

```
>> sff(11,9)
ans =
     1
>> sff(21, 35)
ans =
     7
>>
```

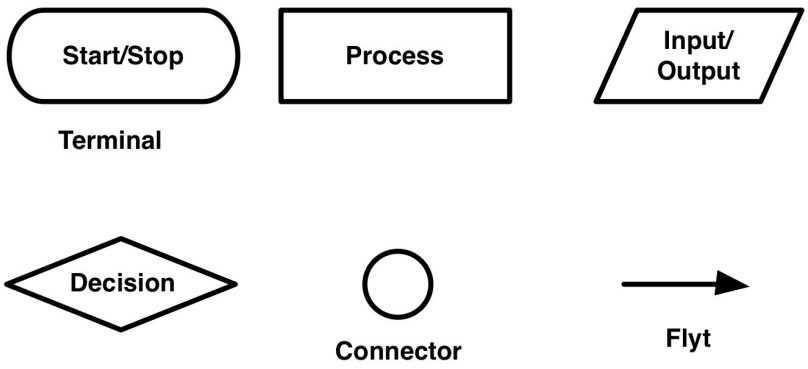
NTNU

Flytskjema

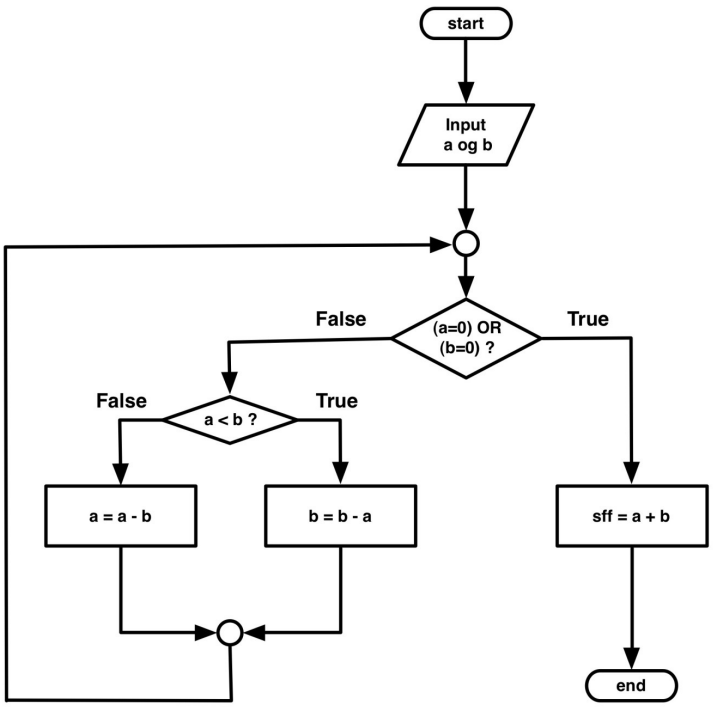
- En **grafisk** representasjon av en algoritme
- Nødvendige steg og kontrollstrukturer for å løse et programmeringsproblem.
- Et sett grafiske elementer
- Fordeler:
  - Grafisk, noe annet enn tekst
- Ulemper:
  - Grafisk: Tar tid, tungt å endre
  - Blir for store
  - Forskjellig fra programmer



Flytskjema: Symboler



sff i flytskjema



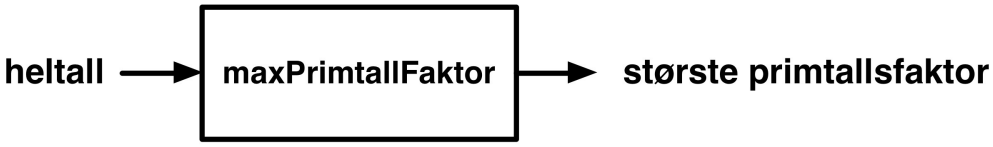
Euklids algoritme, vurdering

- Svært lite effektiv i enkelte tilfeller
  - sff(1, 1000000)
- Finnes mye bedre variant
  - http://no.wikipedia.org/wiki/Euklids_algoritme
  - Trening: Implementer denne selv

35

Største primtallsfaktor i tall

- Alle positive heltall kan faktoriseres i primtallsfaktorer
  - 6 = 3 * 2
  - 99 = 11 * 3 * 3
- Vi skal lage en funksjon maxPrimtallFaktor som finner den største primtallsfaktoren til et heltalltall
- Ide: Fjerner de mindre faktorene til vi står igjen med en faktor, som er den største



NTNU

36

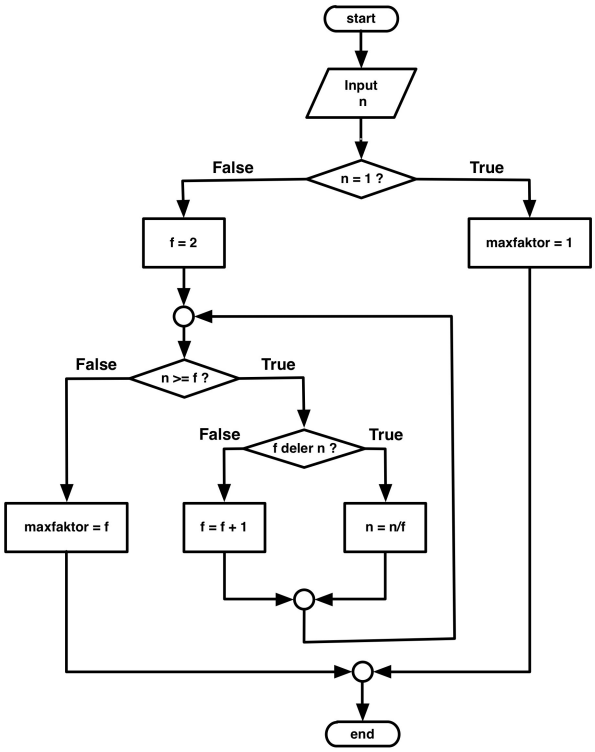
Pseudokode

Input: n

hvis n = 1
  maxfaktor = 1
ellers
  faktor = 2
  gjenta så lenge n >= faktor
    hvis faktor deler n
      % fjerner faktor
      n = n/faktor
    ellers
      % må prøve neste tall
      faktor = faktor + 1
    slutthvis
  sluttgjenta
  maxfaktor = faktor

NTNU

Flytskjema



maxPrimtallFaktor.m

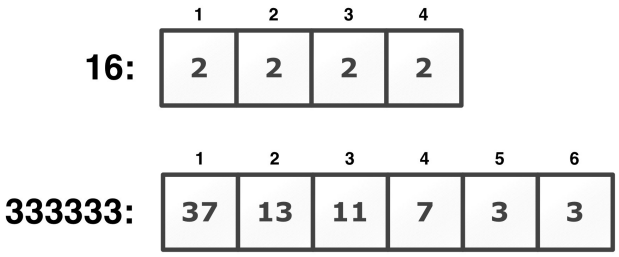
```
function f = maxPrimtallFaktor(n)
% Finner det storste primtallet som er en faktor i n

if n < 1
    % avslutter med feilmelding
    error('Feil i maxPrimtallFaktor: n < 1!')
end %if

if n == 1
    f = 1;
else
    f = 2;
    while n >= f
        if mod(n, f) == 0
            % f er faktor
            n = n/f;
        else
            % f er ikke en faktor (lengre)
            f = f + 1;
        end %if
    end % while
end % if

end % function
```

```
>> maxPrimtallFaktor(99)

ans =

    11

>> maxPrimtallFaktor(333333)

ans =

    37

>> maxPrimtallFaktor(1)

ans =

     1

>>
```

Primtallsfaktorisering

- 33 = 11 * 3
- 16 = 2 * 2 * 2 * 2

- Kan bruke maxPrimtallFaktor som byggekloss
- Lagrer de enkelte faktorene i en vektor




```
function v = primtallFaktorisering(n)
% finner primtallsfaktorene i n

   nesteFaktorNr = 1;

if n == 1
    v(nesteFaktorNr) = 1;
else

   restAvN = n;

   while restAvN > 1
       % tar vare paa den storste (gjenvarende faktoren)
       v(nesteFaktorNr) = maxPrimtallFaktor(restAvN);

       % oppdaterer det som staar igjen av N
       restAvN = restAvN/v(nesteFaktorNr);

       % oppdaterer nr for neste faktor
       nesteFaktorNr = nesteFaktorNr + 1;

   end % while

end % if

end %function
```

Eksempelkjøringer

```
>> primtallFaktorisering(1)

ans =

     1

>> primtallFaktorisering(34)

ans =

    17     2

>> primtallFaktorisering(333333)

ans =

    37    13    11     7     3     3

>> primtallFaktorisering(1000000)

ans =

     5     5     5     5     5     5     2     2     2     2     2     2

>>
```


Problemløsning, VK

http://www.youtube.com/watch?v=IUEvqDwcDYs