



NTNU

Kunnskap for en bedre verden

TDT4105 Informasjonsteknologi, grunnkurs

Matlab 5: Løkker (FOR og WHILE)

Matlab 6: Problemløsning / Algoritmer

Rune Sætre (satre@idi.ntnu.no)

Anders Christensen (anders@idi.ntnu.no)

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
```

- Løkketroppen utføres en gang for hvert element i tabell

Summerer de 10 minste primtallene

```
% 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 et tall

maxTall = 10; % det stoerste tallet

tall = randi(maxTall); % trekker tallet

gjetter = true;
antallForsok = 0;

while gjetter

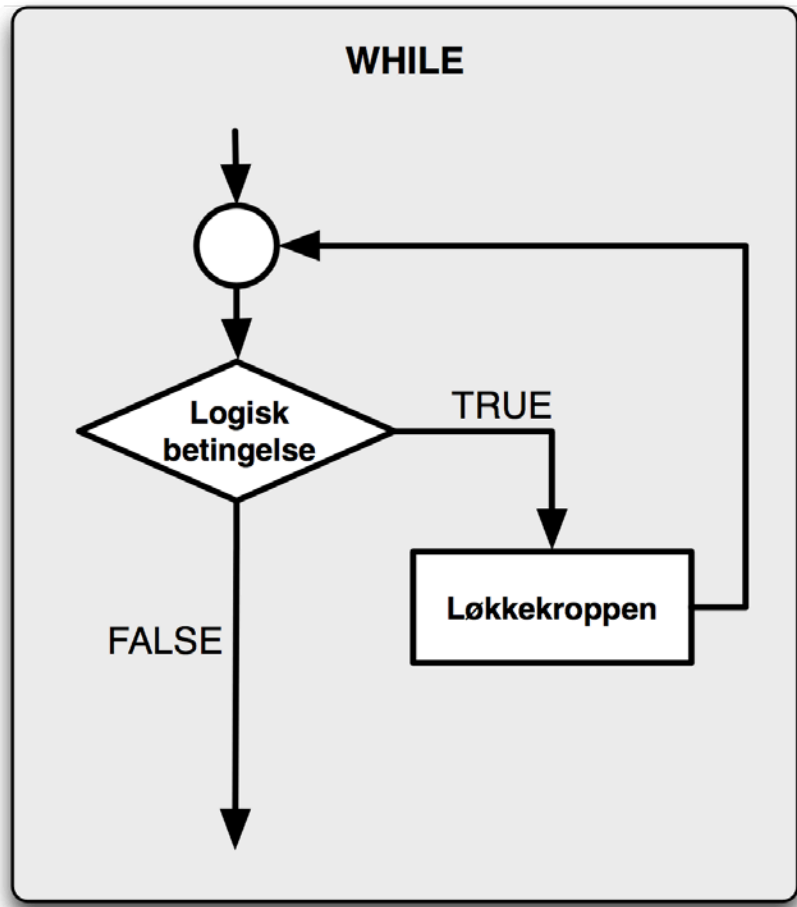
    forsok = input(['Gjett et tall fra 1 til ' num2str(maxTall) ': ']);
    if forsok == tall
        gjetter = false;
    else
        disp('Feil, prøv igjen');
    end
    antallForsok = antallForsok + 1;
end % while

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

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

- 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 estimer.

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   0.000004
0.0100000000: 3.1465677472    0.0049750936   0.000001
0.0010000000: 3.1420924037    0.0004997501   0.000006
0.0001000000: 3.1416426511    0.0000499975   0.000056
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.

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 løkke kroppen
 - MÅ ha et resonnement for terminering etter hvert



NTNU

Det skapende universitet

TDT4105 Informasjonsteknologi grunnkurs:

Uke 39/40 – Algoritmeformulering:

- Flytskjema
- Pseudokode

Anders Christensen

anders@idi.ntnu.no

Rune Sætre

satre@idi.ntnu.no

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).

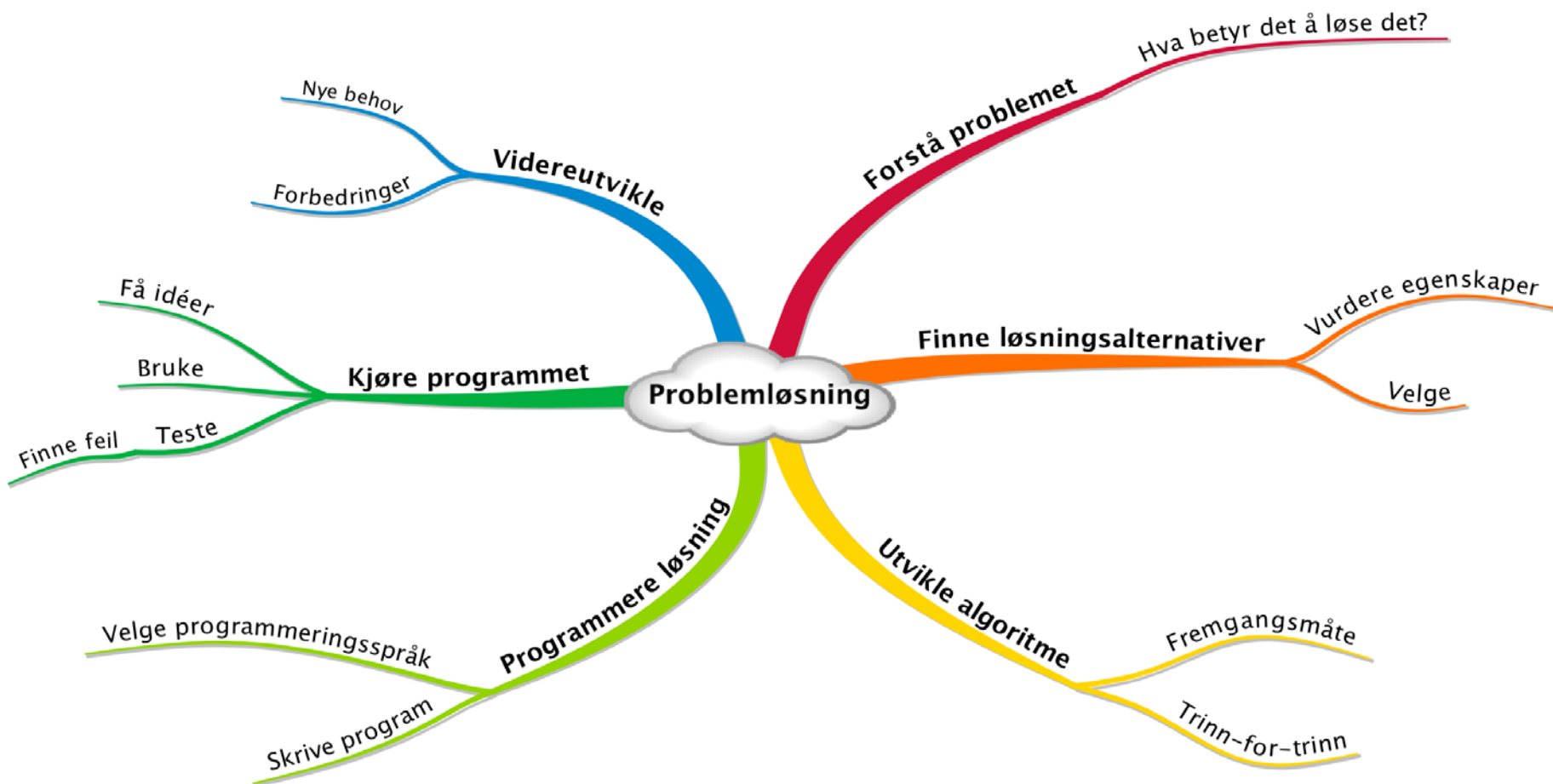
Algoritme

- 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).

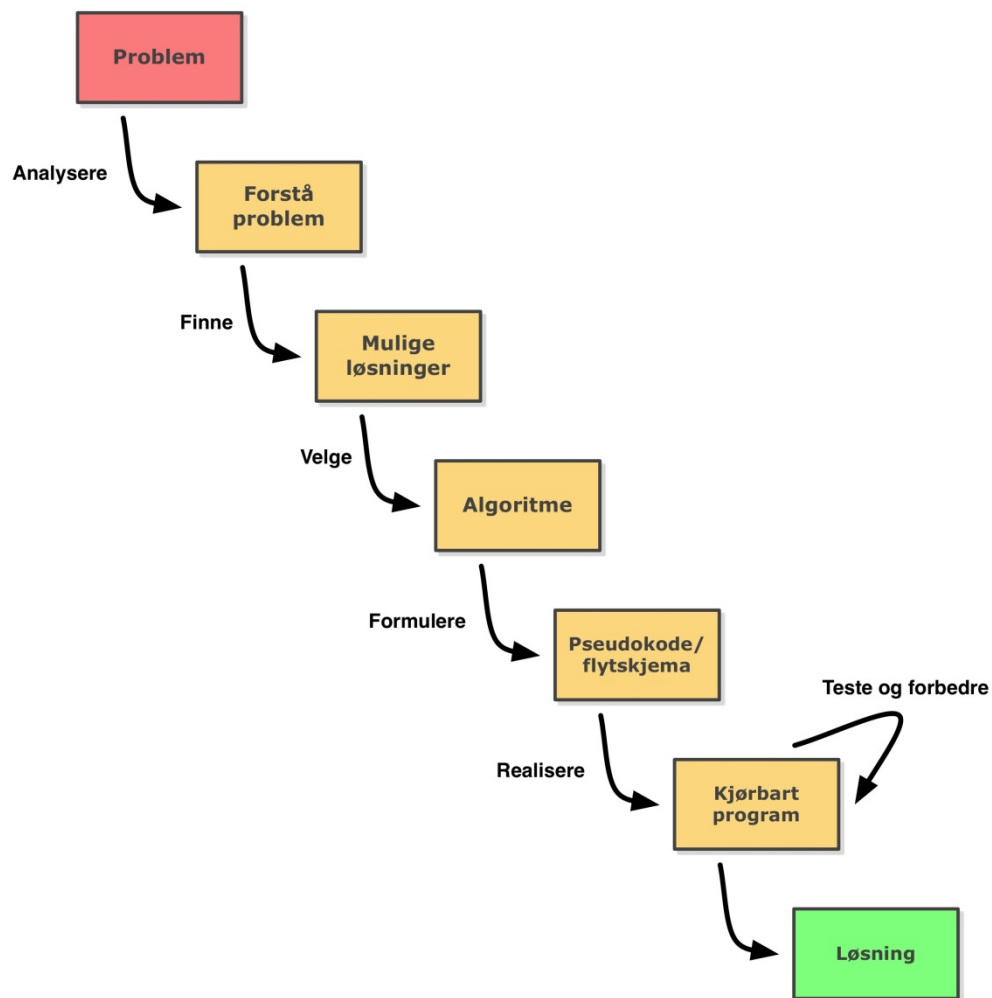
Euklids algoritme

- 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) – $\text{sff}(a,b)$
 - Hvis et av tallene er 0 har vi løsning: $\text{sff}(a,0) = a$
 - Ellers trekker vi det minste tallet fra det største tallet helt til ett av tallene er lik null, det andre tallet er da svaret.
 - $\text{sff}(35,21)=\text{sff}(14,21)=\text{sff}(14,7)=\text{sff}(7,7)=\text{sff}(7,0)$
 - $\text{sff}(9,11)=\text{sff}(9,2)=\text{sff}(7,2)=\dots=\text{sff}(1,2)=\text{sff}(1,1)=\text{sff}(1,0)$
- NB! Finnes mer effektive algoritmer

Problemløsningsprosess



Problemløsningsprosessen



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

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 = $a + b$

sff i Matlab-kode (sff.m)

```
function f = sff(a, b)
% største 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

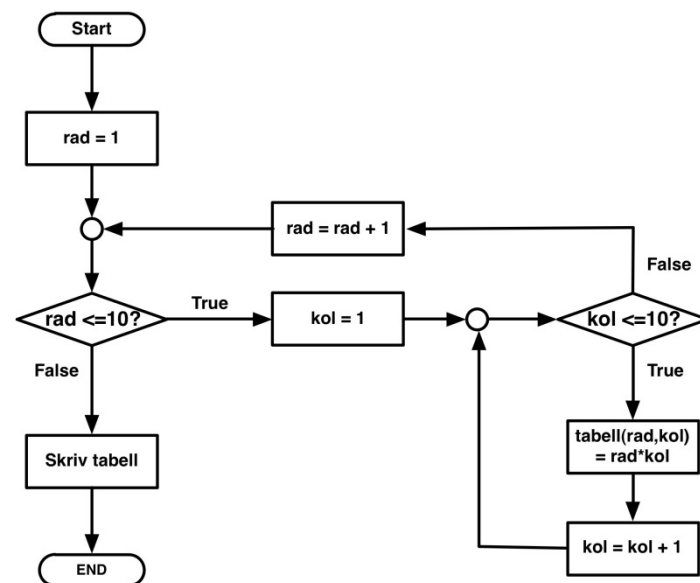
f = a + b;

end % function
```

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

Flytskjema

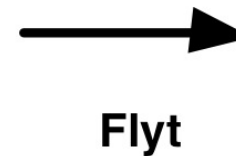
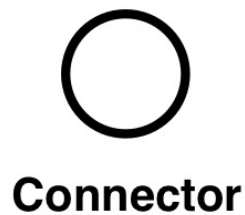
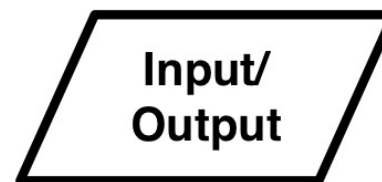
- En **grafisk** representasjon av en algoritme
- Nødvendige steg og kontrollstrukturer for å løse et programmeringsproblem.
- Et sett grafiske elementer
- Fordeler:
 - Grafisk, noe annet enn tekst
- Ulemper:
 - Grafisk: Tar tid, tungt å endre
 - Blir for store
 - Forskjellig fra programmerer



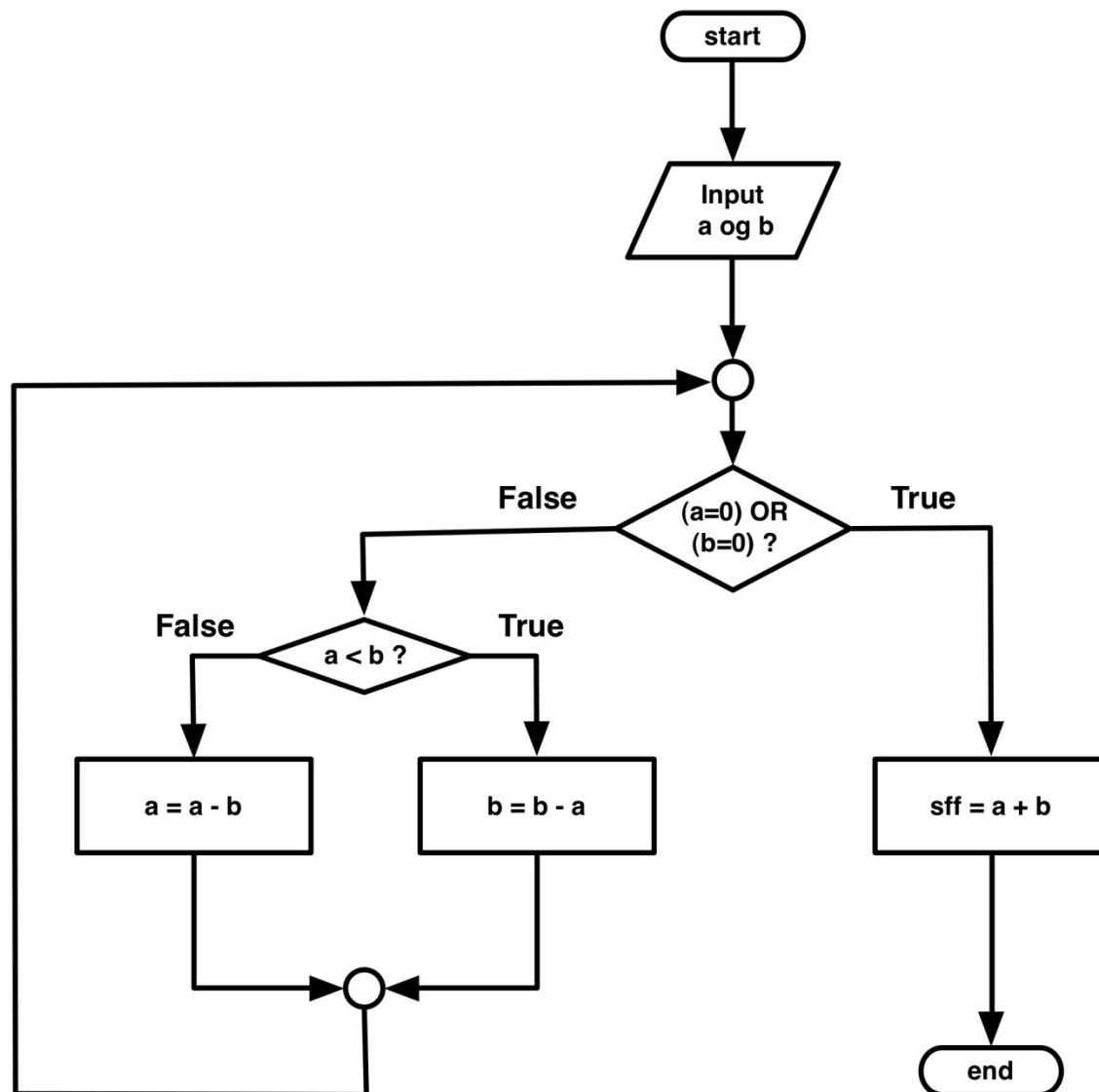
Flytskjema: Symboler



Terminal



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