Introduktion till MATLAB-programmering

- Algoritmer
- Script- och funktionsfiler
- Logiska uttryck
- Villkorssatser: `if`-satser
- Repetitionssatser (loopar): `for`-satser, `while`-satser

Från problem till program

- Anledningen till att skriva ett program, att programmera, är att *få en dator att lösa ett specifikt problem.*
- Programmering och problemlösning har därför mycket gemensamt.

Från problem till program (forts.)

- Moment vid "programskrivning":
  1. *Specificera problemet.*
     (Indata. Resultat/utdata.)
  2. *Analysera problemet.*
     (Strukturera problemet. Dela upp i delproblem.)
  3. *Gör upp en plan för hur problemet/delproblemen skall lösas.*
     (Välj lösningsmetod. Konstruera en algoritm.)
  4. *Koda programmet.*
     (Översätt algoritmen till MATLAB-kommandon. Skriv M-fil.)
  5. *Testa programmet.*
     (Använd olika indata. Vid behov felsök.)
  6. *Dokumentera programmet.*
     (Beskriv problem, algoritmer, indata och utdata.)

Algoritmer

- En algoritm anger de instruktioner (kommandon/satser) som skall utföras för att lösa ett problem eller ett delproblem.
- Man använder sig huvudsakligen av följande tre konstruktioner:
  1. *Sekvens*
     Satser utförs en och en i tur och ordning.
  2. *Selektion/Alternativ*
     Olika satser utförs beroende på om vissa villkor är uppfyllda.
     MATLAB: `if`-sats
  3. *Iteration/Repetition*
     En grupp av satser utförs flera gånger.
     MATLAB: `while`-sats, `for`-sats

Algoritmer (forts.)

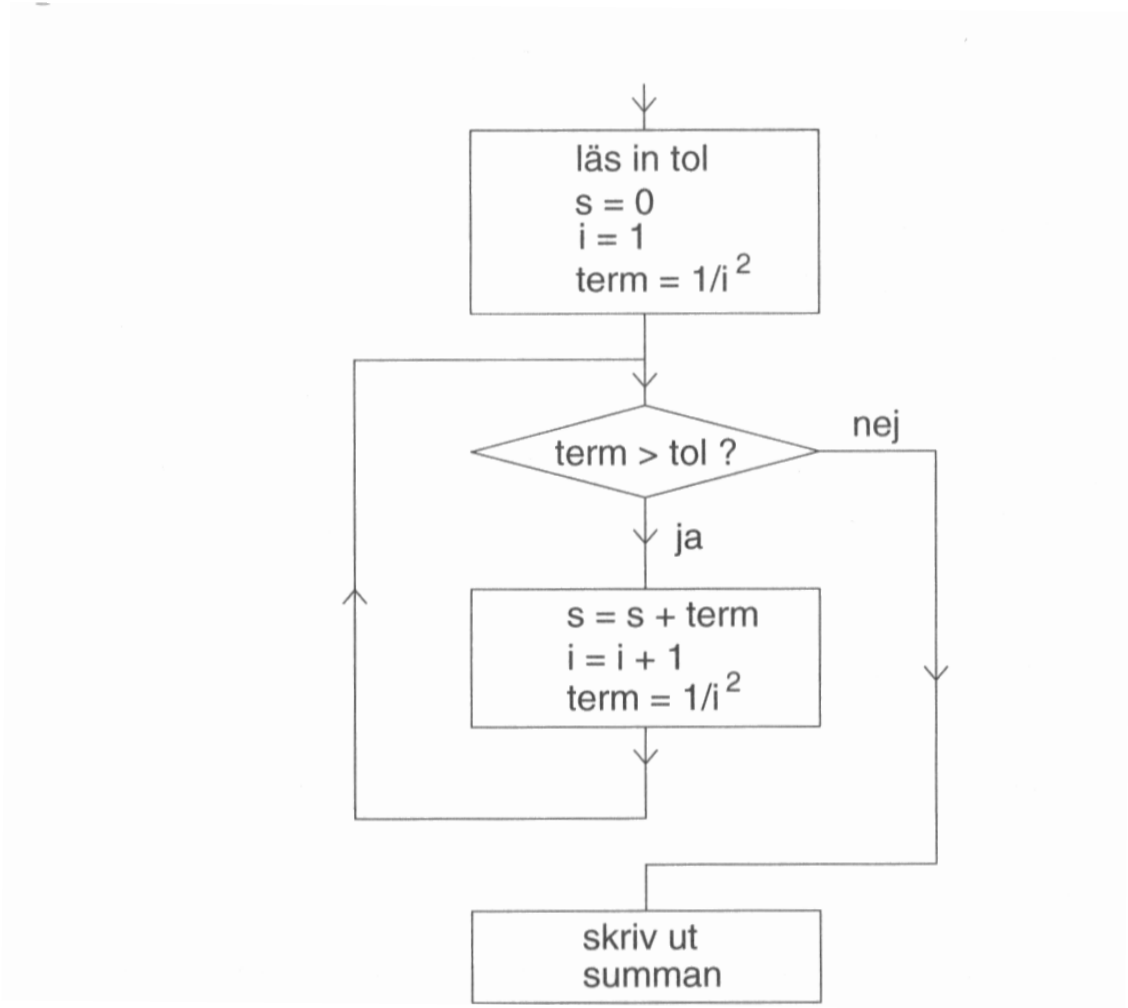
- En algoritm uttrycks ofta med hjälp av ett *flödesschema*.
- **Exempel.** Algoritm för att beräkna en approximation

$$s_N = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{N^2}$$

till summan

$$s = \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \ldots$$

där N väljs så att $1/N^2 > TOL$ men $1/(N+1)^2 \leq TOL$,
där TOL är en given tolerans.


läs in tol
s = 0
i = 1
term = 1/i 2
term > tol ?
nej
ja
s = s + term
i = i + 1
term = 1/i 2
skriv ut
summan

Funktionsfiler

- De M-filer vi skrivit hittills kallas *kommandofiler* eller *scriptfiler*.
- Vi skall nu studera en annan typ av M-filer som kallas *funktionsfiler*.
- En funktionsfil kan ha *inparametrar* och *utparametrar*.
- En funktionsfil kan *anropas* från MATLAB-prompten, från en kommandofil eller från en annan funktionsfil.

Funktionsfiler. Anrop av funktionsfiler.

- Första raden i en funktionsfil skall alltid inledas med ordet `function`.
- *Exempel.* Vi skapar en funktionsfil **funk1.m** som beräknar $y = x^2 + 1$.

```
function y = funk1(x)
% Denna funktionsfil beräknar y = x^2 + 1.
% Inparameter: x
% Utparameter: y
y = x.^2 + 1;  % Elementvis potens så kan inparameter
               % också vara en vektor.
```

Funktionsfiler. Anrop av funktionsfiler. (forts.)

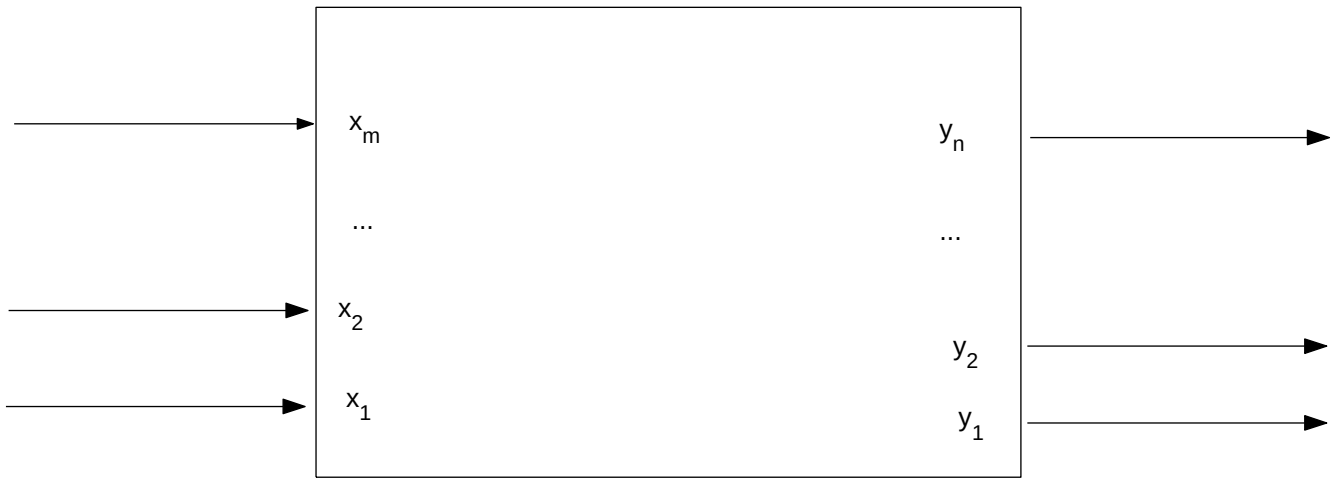
- *Exempel. (forts.)* Några *anrop* av **funk1** från MATLAB-prompten:

```
>> funk1(0)
ans =
     1

>> y1 = funk1(1)
y1 =
     2

>> x2 = 2; y2 = funk1(x2 + 1)
y2 =
     10

>> x = [0 1 2]; y = funk1(x)
y =
     1     2     5
```

Funktionsfiler. Anrop av funktionsfiler. (forts.)

- *Syntax.* Funktionsfilen **filnamn.m** med inparametrar $x_1, x_2, ..., x_m$ och utparametrar $y_1, y_2, ..., y_n$ skall se ut på följande sätt:

```
function [y1,y2,...,yn] = filnamn(x1,x2,...,xm)
% Kommentarer.
%    Beskrivning av funktionen samt av in- och utparametrar.
% Satser.
%    I dessa satser måste y1,y2,...,yn tilldelas värden.
```



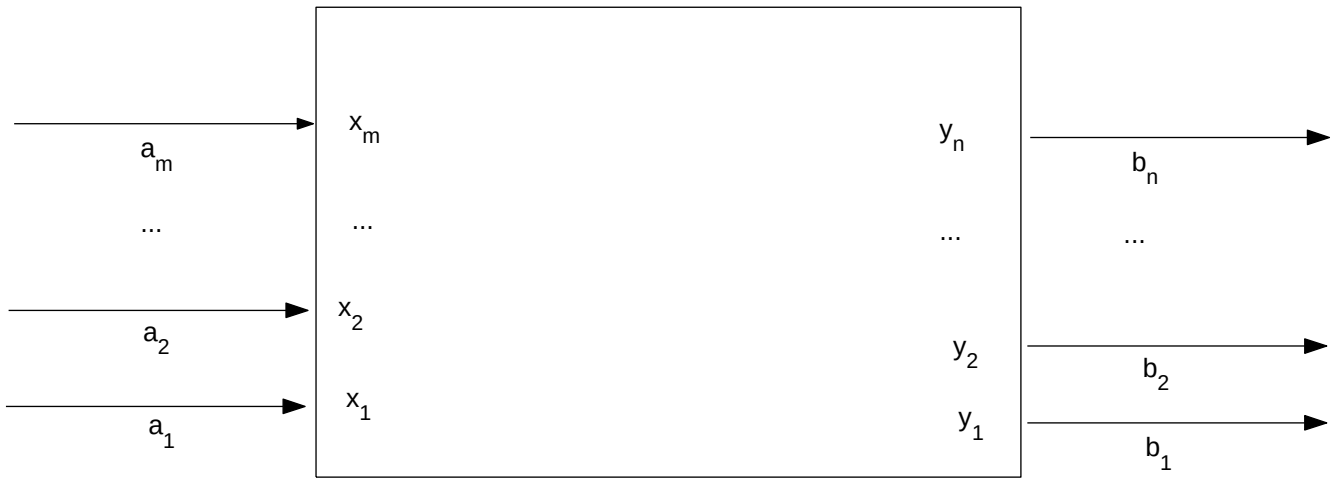
- Funktionen kan liknas vid en maskin, där $x_1, x_2, ..., x_m$ ges värden från input.
- Funktionen producerar ett resultat $y_1, y_2, ..., y_n$ som sedan skickas vidare (output).

Funktionsfiler. Anrop av funktionsfiler. (forts.)

- För att anropa funktionsfilen **filnamn.m** skriver man:

```
[b1,b2,...,bn] = filnamn(a1,a2,...,am)
```

- Inparametrarna $a_1, a_2, ..., a_m$ kallas ibland *aktuella parametrar*.
- Motsvarande parametrar $x_1, x_2, ..., x_m$ inuti funktionsfilen kallas *formella parametrar*.
- Vid funktionsanropet kopieras de aktuella parametrarna $(a_1, a_2, ..., a_m)$ till motsvarande formella parametrar $(x_1, x_2, ..., x_m)$ i funktionsfilen.
- När funktionen är färdigexekverad kopieras utparametrarna $y_1, y_2, ..., y_n$ till motsvarande variabler $b_1, b_2, ..., b_n$.
- OBS! Därefter *raderas* alla parametrar och lokala variabler (d.v.s. sådana som skapats inuti funktionsfilen) i funktionsfilen och ligger alltså inte kvar i minnet.



- OBS! De aktuella parametrarna påverkas inte av vad som händer inuti funktionsfilen, t.ex. om de formella parametrarnas värden ändras.

Funktionsfiler. Anrop av funktionsfiler. (forts.)

- *Exempel.* Funktionsfilen **medelv.m** beräknar medelvärdet av två tal x och y.

```
function z = medelv(x,y)
% Denna funktionsfil beräknar medelvärdet
% z = (x+y)/2 av talen x och y.
% Inparametrar: x,y
% Utparameter: z
z = (x+y)/2;
```

Funktionsfiler. Anrop av funktionsfiler. (forts.)

- *Exempel. (forts.)* Ett par *anrop* av **medelv** från MATLAB-prompten:

```
>> a = 4; b = 7; c = medelv(a,b)
c =
    5.5000

>> medelv(4,6)   % Anges ingen variabel hamnar resultatet i ans.
ans =
     5
```

Logiska uttryck

- Ett *logiskt uttryck* är ett uttryck som kan vara antingen *sant* eller *falskt*.
- *Exempel. Sanna* logiska uttryck.

  "En kvadrat har fyra sidor."

  "MATLAB betyder MATrix LABoratory."

  "5 är större än 4."
- *Exempel. Falska* logiska uttryck.

  "En triangel har fyra sidor."

  "Julafton infaller i februari."

  "5 är lika med 4."
- I de understa exemplen har vi *jämfört* talen 4 och 5.

Logiska uttryck (forts.)

- I MATLAB finns följande *relationsoperatorer* (jämförelseoperatorer), som kan användas när man vill göra jämförelser:

Relationsoperator	Betydelse
<	mindre än
<=	mindre än eller lika med
>	större än
>=	större än eller lika med
==	lika med
~=	inte lika med

Logiska uttryck (forts.)

- *Exempel. Sanna* logiska uttryck i MATLAB.

```
2 < 5
4 <= 5
8 > 5
7 >= 7
3 == 3
3 ~= 4
```

- *Exempel. Falska* logiska uttryck i MATLAB.

```
1 < 0
6 <= 2
2 > 6
1 >= 2
3 == 5
3 ~= 3
```

Logiska uttryck (forts.)

- Ett logiskt uttryck i MATLAB får värdet **1** om uttrycket är *sant*, och värdet **0** om uttrycket är *falskt*.
- *Exempel.*

```
>> 2 < 5
ans =
   1

>> 3 == 3
ans =
    1

>> 2 >= 6
ans =
   0

>> 3 ~= 3
ans =
   0
```

Logiska uttryck (forts.)

- *Exempel.*

```
>> a = 3; b = 4;

>> a > 2    % Sant.
ans =
     1

>> b <= 0   % Falskt.
ans =
     0

>> a < b    % Sant.
ans =
     1

>> a = 5;

>> a < b    % Falskt.
ans =
     0
```

Logiska uttryck (forts.)

- *Exempel.* Skilj noga på *relationsoperatorn* == och *tilldelning* av värde till variabel som görs med =

```
>> x = 7     % x tilldelas värdet 7.
x =
     7

>> x == 3    % Falskt logiskt uttryck.
ans =
     0

>> x = 3     % x tilldelas värdet 3.
x =
     3

>> x == 3    % Sant logiskt uttryck.
ans =
     1
```

Logiska uttryck (forts.)

- Logiska uttryck kan kombineras med *logiska operatorer*:

Logisk operator	Betydelse
&	logiskt **och**
|	logiskt **eller**
~	logiskt **inte** (logisk **negation**)

Logiska uttryck (forts.)

- Låt A och B vara två logiska uttryck.
- Det logiska uttrycket

  $$A \,\&\, B \qquad \text{“A och B”}$$

  är **sant** om A är **sant** och B är **sant**.
- Det logiska uttrycket

  $$A \mid B \qquad \text{“A eller B”}$$

  är **sant** om A är **sant** och/eller B är **sant**.
- Det logiska uttrycket

  $$\sim A \qquad \text{“inte A”}$$

  är **sant** om A är **falskt**, och **falskt** om A är **sant**.

Logiska uttryck (forts.)

- *Exempel.*

```
>> a = 3; b = 4;

>> a<10 & b>0   % Sant uttryck.
ans =
     1

>> a<10 & b<0   % Falskt uttryck.
ans =
     0

>> a<10 | b<0   % Sant uttryck.
ans =
     1

>> a==10 | b<0   % Falskt uttryck.
ans =
     0
```

Logiska uttryck (forts.)

- De logiska operatorerna & och | har *lägre prioritet* än relationsoperatorerna (<, >, et.c.), vilka i sin tur har *lägre prioritet* än de aritmetiska operatorerna (+, -, *, /).
- Den logiska operatorn ~ har *högre prioritet* än de aritmetiska operatorerna, men *lägre prioritet* än potenser.

Logiska uttryck (forts.)

- *Prioritetstabell*

  1. `Parenteser ()`
  2. `Potens (^)`
  3. `Negation (-), Logisk negation (~)`
  4. `Multiplikation (*), Division (/)`
  5. `Addition (+), Subtraktion (-)`
  6. `Kolonoperatorn (:)`
  7. `Mindre än (<), Mindre än eller lika med (<=)`
     `Större än (>), Större än eller lika med (>=)`
     `Lika med (==), Inte lika med (~=)`
  8. `Logiskt OCH (&)`
  9. `Logiskt ELLER (|)`

- Är du osäker: *Använd parenteser.*

Logiska uttryck (forts.)

- *Exempel.*

```
>> 1+1 < 4 & 2*3 > 5
ans =
     1
```

är samma sak som:

```
>> ((1+1) < 4) & ((2*3) > 5)
ans =
     1
```

If-satser

- Vi skall nu se hur logiska uttryck kan utnyttjas i programmering.
- Beroende på om ett visst logiskt uttryck är **sant** eller **falskt** kan programmet instrueras att göra olika saker.
- Detta kallas *selektion* (eller *alternativ*), d.v.s. val mellan två eller flera olika "vägar" i programmet.

If-satser (forts.)

- Det vanligaste sättet att åstadkomma selektion i ett program är att använda `if`-satsen.
- *Exempel.* Antag att vi skrivit M-filen **exempel1.m**:

```
a = input('Mata in ett positivt tal: ');
if a<=0
  disp('Du matade in fel!')
end
disp('Du matade in talet:')
disp(a)
```

If-satser (forts.)

- *Exempel. (forts.)* Följande utskrift fås vid två olika körningar:

```
>> exempel1
Mata in ett positivt tal: 20
Du matade in talet:
    20

>> exempel1
Mata in ett positivt tal: -12
Du matade in fel!
Du matade in talet:
   -12
```

- Notera att kommandot

```
disp('Du matade in fel!')
```

endast utförs *om* det logiska uttrycket `a<=0` är **sant**.

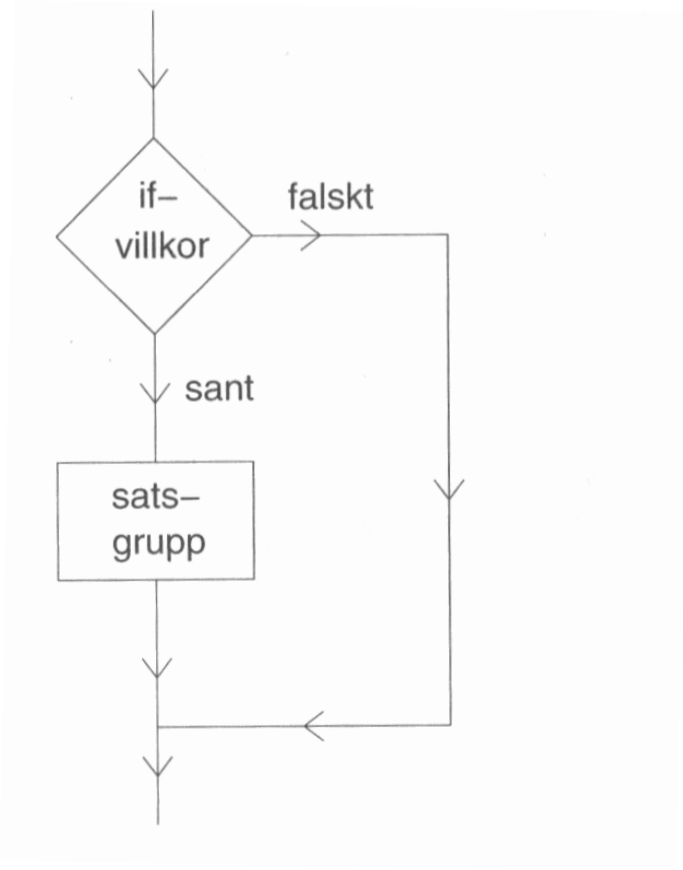
If-satser (forts.)

- *Syntax.* (Enklaste formen av `if`-sats.)

  `if` *logiskt uttryck*
  kommando 1
  kommando 2
  ...
  `end`

- Kommandona mellan `if` och `end` utförs bara om *logiskt uttryck* är **sant**.

- Notera att kommandona mellan `if` och `end` "skjutits in" lite till höger. Detta kallas att *indentera*. Det spelar ingen roll för MATLAB:s del men det ger *läsliga program*.



Flödesschema för if-sats.

If-satser (forts.)

- Vi skall nu se på en mer generell form av `if`-satsen.
- *Exempel.* Antag att vi skrivit M-filen **exempel2.m**:

```
a = input('Mata in ett tal: ');
if a<0
  disp('Du matade in ett negativt tal!')
  a = -a;
elseif a==0
  disp('Du matade in talet 0!')
else
  disp('Du matade in ett positivt tal!')
end
disp('Absolutbeloppet av talet är:')
disp(a)
```

If-satser (forts.)

- *Exempel. (forts.)* Följande utskrift fås vid tre olika körningar:

```
>> exempel2
Mata in ett tal: 3
Du matade in ett positivt tal!
Absolutbeloppet av talet är:
     3

>> exempel2
Mata in ett tal: 0
Du matade in talet 0!
Absolutbeloppet av talet är:
     0

>> exempel2
Mata in ett tal: -2
Du matade in ett negativt tal!
Absolutbeloppet av talet är:
     2
```

If-satser (forts.)

- Vi kan också skriva en funktionsfil `min_abs.m`:

```
function y = min_abs(x)
% Denna funktion beräknar absolutbeloppet av x
% enligt regeln:
%
%  min_abs(x) = x, om x >= 0
%               -x, om x < 0
if x >= 0
    y = x;
else
    y = -x;
end
```

If-satser (forts.)

- Testkörning:

```
>> min_abs(3)
ans =
     3

>> min_abs(0)
ans =
     0

>> min_abs(-3)
ans =
     3
```

If-satser (forts.)

- *Syntax.*

  `if` *logiskt uttryck 1*

  satsgrupp 1

  `elseif` *logiskt uttryck 2*

  satsgrupp 2

  ...

  `elseif` *logiskt uttryck n*

  satsgrupp n

  `else`

  satsgrupp

  `end`

- En *satsgrupp* står för en eller flera satser/kommandon.
- Notera att man, liksom i den enklaste formen av `if`-satsen tidigare, kan utelämna `elseif` och/eller `else`.

If-satser (forts.)

- *Syntax. (forts.)*

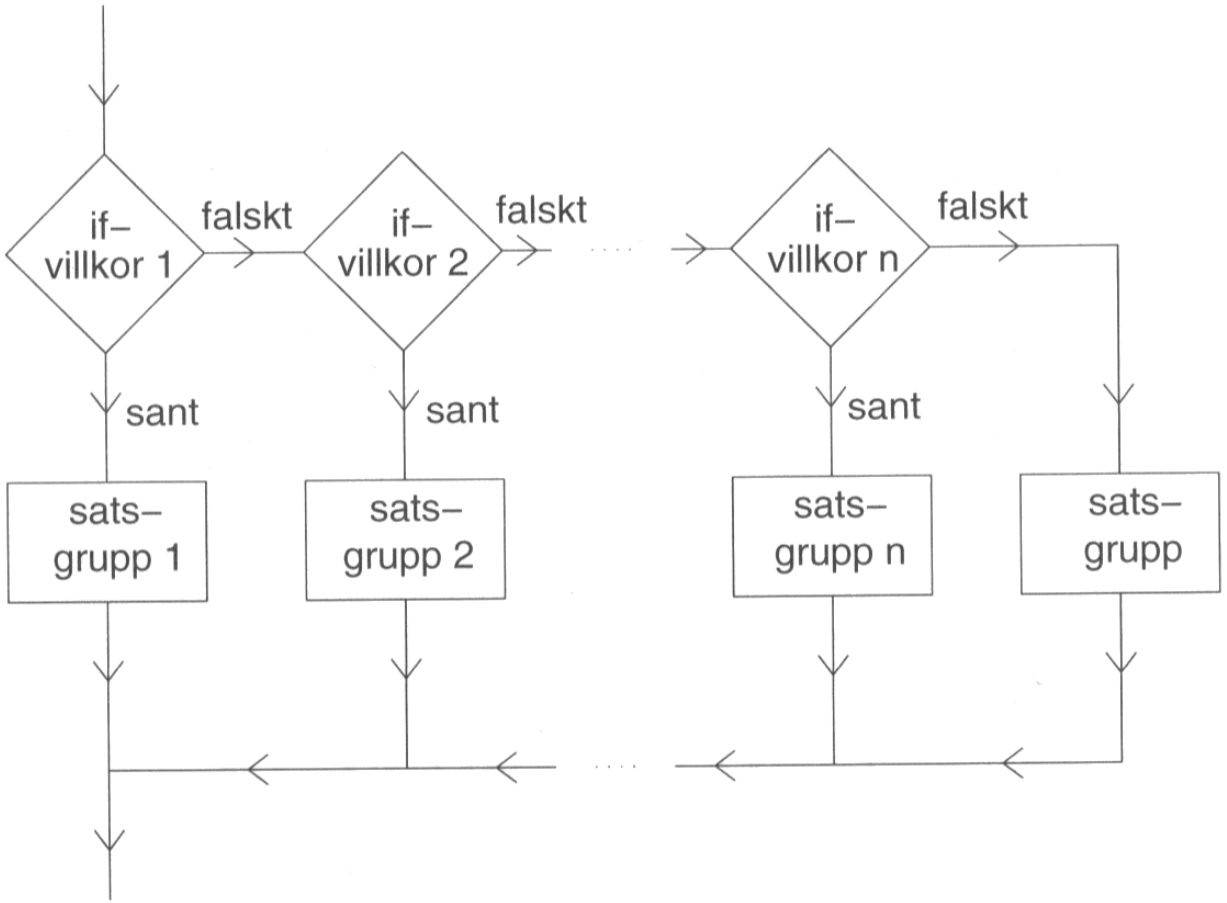
  Kommandona i *satsgrupp 1* utförs om *logiskt uttryck 1* är **sant**.

  Kommandona i *satsgrupp 2* utförs om *logiskt uttryck 1* är **falskt** och *logiskt uttryck 2* är **sant**.

  ...

  Kommandona i *satsgrupp n* utförs om *logiskt uttryck 1* t.o.m. *logiskt uttryck n-1* är **falska** och *logiskt uttryck n* är **sant**.

  Kommandona i *satsgrupp* utförs om samtliga logiska uttryck är **falska**.



Flödesschema för if-sats.

Repetitionssatser

- I den enklaste typen av program utförs *samtliga* kommandon (i tur och ordning) *precis en gång*.
- Vi har nyss sett hur man med `if`-satsen kan få programmet att *välja* att utföra vissa kommandon, men inte andra. Detta kallas *selektion* (*alternativ*).
- Vi skall nu se hur man med `while`-satsen och `for`-satsen kan få programmet att utföra vissa kommandon *flera gånger*. Detta kallas *iteration* (*repetition*).

While-loopar

- I en while-loop repeteras en satsgrupp så länge som ett logiskt uttryck är sant.
- *Syntax.*

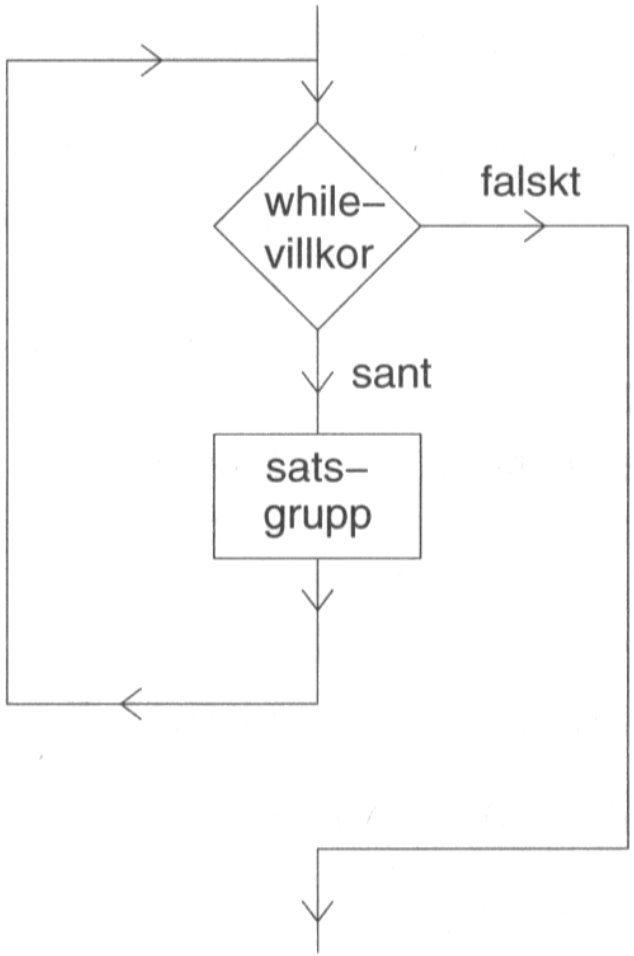
  `while` *logiskt uttryck*

  sats 1

  sats 2

  ...

  `end`
- Satserna mellan `while` och `end` utförs *så länge* som *logiskt uttryck* är **sant**.



Flödesschema för en while-loop.

While-loopar (forts.)

- *Exempel.* Skriv ett program som ber användaren mata in ett positivt tal. Om talet som matas in är negativt skall användaren uppmanas att mata in ett nytt tal, *så länge* som inmatningen är felaktig. Därefter skall kvadratroten ur talet beräknas och skrivas ut på skärmen.

  Vi löser problemet med en `while`-sats i M-filen **kvadratrot.m**:

```
x = input('Mata in ett positivt tal: ');
while x<0
  x = input('FEL!!! Mata in ett POSITIVT tal: ');
end
disp('Roten ur talet är:')
disp(sqrt(x))
```

While-loopar (forts.)

- *Exempel. (forts.)* En körning av programmet **kvadratrot** ger följande resultat:

```
>> kvadratrot
Mata in ett positivt tal: -2
FEL!!! Mata in ett POSITIVT tal: -4
FEL!!! Mata in ett POSITIVT tal: -1
FEL!!! Mata in ett POSITIVT tal: 16
Roten ur talet är:
     4
```

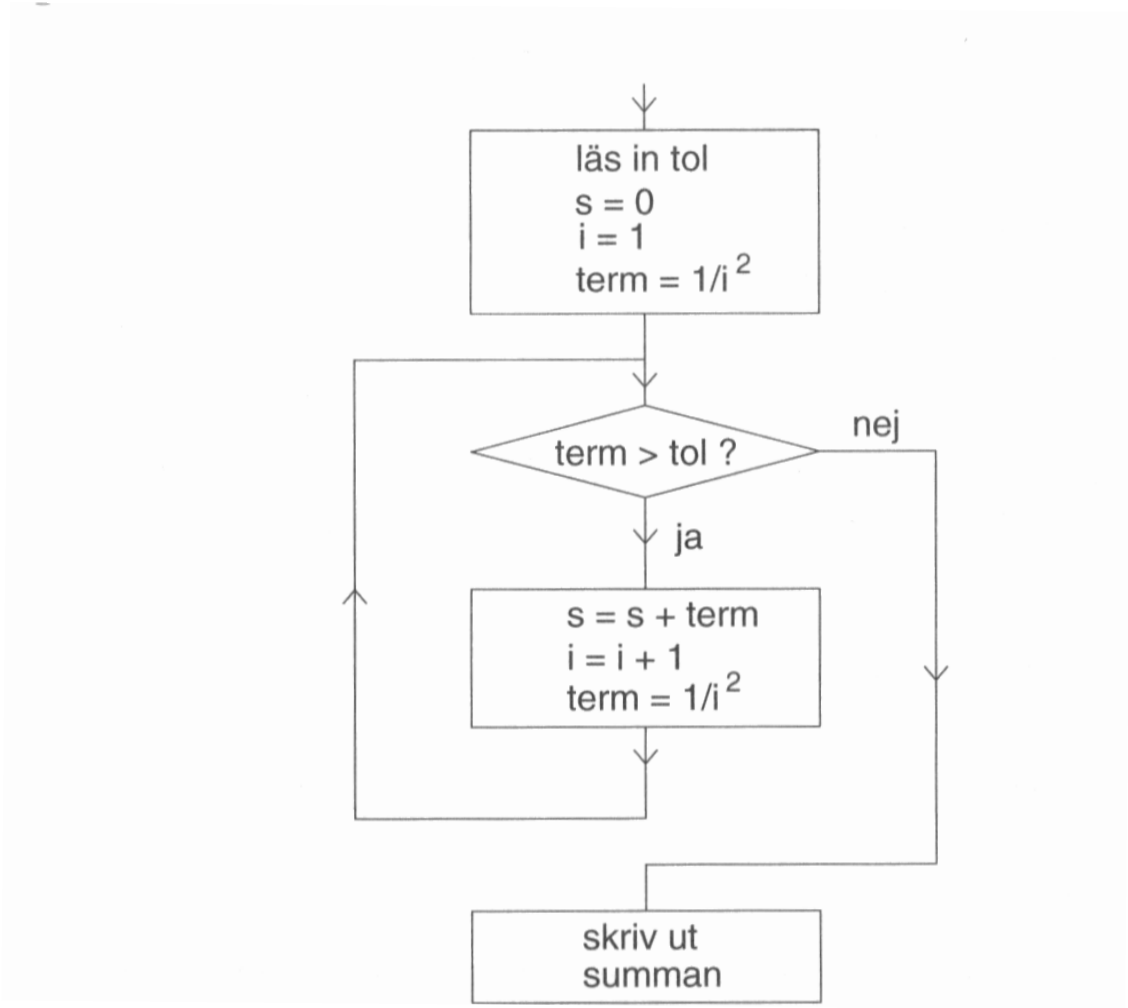
While-loopar (forts.)

- *Exempel.* Skriv ett program som först ber användaren mata in en tolerans TOL. Därefter skall programmet beräkna summan

$$s_N = \sum_{i=1}^{N} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{N^2}$$

  där N väljs så att $1/N^2 > TOL$ men $1/(N+1)^2 \leq TOL$.

  Termer skall alltså adderas *så länge* som de är större än TOL.


läs in tol
s = 0
i = 1
term = 1/i 2
term > tol ?
nej
ja
s = s + term
i = i + 1
term = 1/i 2
skriv ut
summan

While-loopar (forts.)

- *Exempel. (forts.)* Vi löser problemet med en `while`-sats i M-filen **summa.m**:

```
TOL = input('Ge värde på TOL: ');   % Läs in tolerans.
s = 0;                              % Sätt summan till noll.
i = 1;                              % Sätt i till 1.
term = 1/i^2;                       % Beräkna första termen.
while term > TOL                    % Iterera så länge term > TOL.
    s = s + term;                   % Addera term till summan.
    i = i + 1;                      % Stega upp i ett steg.
    term = 1/i^2;                   % Beräkna ny term.
end
fprintf('Summan är %12.10f \n',s)   % Skriv ut resultatet.
fprintf('Det blev N=%u termer',i-1) % Skriv ut N.
```

While-loopar (forts.)

- *Exempel. (forts.)* En körning av programmet **summa** ger följande resultat:

```
>> summa
Ge värde på TOL: 1e-10
Summan är 1.6449240668
Det blev N=99999 termer
```

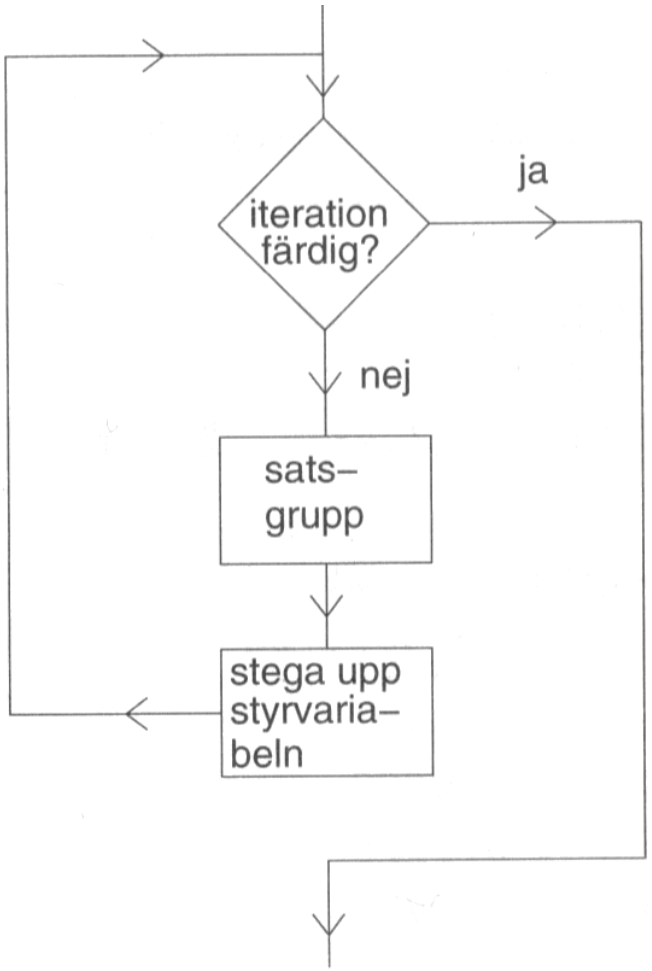
- *Kommentar.* $\lim_{N\to\infty} s_N = \pi^2/6 = 1.64493406684 82...$

For-loopar

- I en for-loop repeteras en satsgrupp så länge en styrvariabel löper i en styrmängd.
- *Syntax.*

  `for` *styrvariabel* = *styrmängd*
  satsgrupp
  `end`
- 1. Styrvariabeln sätts till det *första* värdet i styrmängden och satserna i *satsgrupp* utförs.
  2. Styrvariabeln sätts till det *andra* värdet i styrmängden och satserna i *satsgrupp* utförs.
  ...

  och så vidare tills det att styrvariabeln har löpt igenom hela styrmängden.



Flödesschema för en for-loop.

For-loopar (forts.)

- *Exempel.* För att beräkna och skriva ut kvadraterna på de fem första positiva heltalen kan vi använda en `for`-sats i M-filen **kvadrater.m**:

```
disp('   k     k^2')
disp('------------')
for k = [1 2 3 4 5]
    fprintf('   %u     %u  \n', k, k^2)
end
```

  1. Först tilldelas styrvariabeln k värdet 1. Satsen `fprintf` skriver ut k och k^2.
  2. Sedan tilldelas styrvariabeln k värdet 2. Satsen `fprintf` skriver ut k och k^2.

     ...

     O.S.V.

For-loopar (forts.)

- *Exempel. (forts.)* En körning av programmet **kvadrater** ger följande resultat:

```
>> kvadrater
   k     k^2
------------
   1     1
   2     4
   3     9
   4     16
   5     25
```

For-loopar (forts.)

- Ofta används *kolonnotation* för att definiera styrmängden.
- *Exempel.* För att beräkna den aritmetiska summan

$$s = 5 + 8 + 11 + 14 + \ldots + 101$$

kan vi använda en `for`-sats i M-filen **aritmetisk.m**:

```
summa = 0;       % Sätt summan till noll.
for i = 5:3:101 % Styrvariabeln i löper genom värdena 5,8,11,...,101
    summa = summa + i;             % Addera i till summan.
end
fprintf('Summan är %u. \n',summa) % Skriv ut resultatet.
```

For-loopar (forts.)

- *Exempel. (forts.)* En körning av programmet **aritmetisk** ger följande resultat:

```
>> aritmetisk
Summan är 1749.
```

- *Kommentar.* Två alternativa lösningar utan `for`-loop:

```
>> s = sum(5:3:101)    % Använd MATLAB:s inbyggda summeringsfunktion.
s =
        1749

>> antal_termer = length(5:3:101)
antal_termer =
    33
>> s = 33*(5 + 101)/2  % Formel för beräkning av aritmetisk summa:
s =                    % Summa = antal termer * medelvärdet av
        1749           %           första och sista termen
```