# F2: Styrstrukturer, programmeringsteknik (kap. 3–4)

Villkorskommandon, if, switch
Villkor, logiska värden, relationsoperatorer, logiska operatorer
Skottårsexempel
Triangelanalysexempel, strängar
Switch, tärningsexempel
Felhantering, try-catch, error
Repetitionskommando, slinga, for, while, break, continue
Kuggväxelexempel, blandad utskrift med disp
Datorpecision, eps
Programmeringsstrategi – från problem till program, programskiss (pseudokod ≠ p-kod), algoritm
Filläsning- och skrivning, fopen, fscanf, fprintf



# Villkor if, switch

Man vill att vissa kommandon ska exekveras endast om ett visst villkor är sant.

```
if b>a  % Gör a till den största och b till den minsta av a och b
  temp = a; a = b; b = temp;
end
if villkor
  kommandon
end
```

Man kan vilja exekvera olika kommandon om villkoret är sant resp. falskt

```
if villkor
  kommandon1
else
  kommandon2
end
```



# If-kommando forts.

Ett **if**-kommando kan innehålla ett annat.

```
if villkor1
   kommandon1
   if villkor2
      kommandon2
   end
else
   if villkor3
      kommandon3
   else
      kommandon4
   end
   kommandon5
end
```



# If-kommando forts.

Det finns precis ett `end` till varje `if`.

När några villkor ska undersökas i följd kan man använda `elseif`

```
if villkor1
   kommandon1
elseif villkor2
   kommandon2
else
   kommandon3
end
```

```
if villkor1
   kommandon1
else
   if villkor2
      kommandon2
   else
      kommandon3
   end
end
```

# If-kommando forts.

Det finns inget `end` till ett `elseif`. Man kan utelämna `else`-delen:

```
if villkor1
   kommandon1
elseif villkor2
   kommandon2
end
```

Man kan fortsätta med `else`, `elseif` eller `end` på raden föregånget av »,» eller »;».

```
if b>a; temp = a; a = b; b = temp; end
```



# Villkor

Ett godtyckligt uttryck kan användas som villkor. 0 är falskt. Skilt från 0 är sant. Sant lagras som 1. Ett uttrycks anses sant när realdelen av alla element är nollskilda.

Relationsoperatorerna `<`, `<=`, `>`, `>=`, `==` och `~=` ger logiskt värde som resultat. Logiska värden kan kombineras med de logiska operatorerna `~`, `&`, `|`, `&&` och `||` samt den logiska funktionen `xor`. Det finns många inbyggda funktioner som ger ett logiskt värde som resultat.

```
~a     a&b        a|b        xor(a,b)
a      a b 0 1    a b 0 1    a b 0 1
0 1    0   0 0    0   0 1    0   0 1
1 0    1   0 1    1   1 1    1   1 0
```



# Villkor forts.

Värdet av *k* ligger mellan 5 och 17 skulle man skriva 5<*k*<17 i matematiken. I Matlab skriver man det som `5<k & k<17` dvs. *k* är större än 5 och *k* är mindre än 17. I Matlab har `5<k<17` annan tolkning:

```
k = 1; 5<k<17
ans = 1
```

Först beräknas `5<k` dvs. `5<1` som ju är falskt dvs. `0`. Därefter beräknas `0<17` som ju är sant dvs. `1`.

Det är lätt att skriva fel och förväxla `=` (tilldelningsoperatorn) och `==` (likhetsoperatorn). Operander beräknas före operatorn.
Genvägsoperatorerna `&&` och `||` fungerar annorlunda:

```
% c = a && b              % c = a||b
if ~a c = 0               if a c = 1
else c = b                else c = b
```



# Skottår

Enligt den julianska (romerska) kalendern är ett år skottår om årtalet är delbart med 4. Enligt den gregorianska, i Sverige sedan 1753 använda, kalendern är endast de jämna århundraden som är jämnt delbara med 400 skottår.

Låt `y` vara ett årtal. Sätt `leapjul` till sant om `y` är skottår enligt den julianska och `leapgreg` om `y` är skottår enligt den gregorianska kalendern. Året är ung. 365,2422 dagar och den gregorianska kalendern ger 365,2425 dagar dvs. en dag fel på 3236 år.

```
leapjul = mod(y,4)==0;
y:         1996 2000 2001 2004 2096 2100 2200 2300 2400 2500
leapjul(y):   1    1    0    1    1    1    1    1    1    1
```

Vi vill ha:

```
leapgreg(y):  1    1    0    1    1    0    0    0    1    0
```

Delbart med 4 men inte jämnt århundrade som inte är jämt delbart med 400.

Delbart med 4 och inte (delbart med 100 och inte delbart med 400).

```
leapgreg = leapjul & ~(mod(y,100)==0 & ...
mod(y,400)~=0);
% alternativt kommando
leapgreg = leapjul & mod(y,100)~=0 | ...
mod(y,400)==0;
y:              1996 2000 2001 2004 2096 2100 2200 2300 2400 2500
mod(y, 4)==0:      1    1    0    1    1    1    1    1    1    1
```

Delbara med 100.

```
mod(y, 100)==0:    0    1    0    0    0    1    1    1    1    1
```

Inte delbara med 400.

```
mod(y, 400)~=0:    1    0    1    1    1    1    1    1    0    1
```

Delbara med 100 men inte delbara med 400.

```
mod(y, 100)==0
& mod(y, 400)~=0:  0    0    0    0    0    1    1    1    0    1
```

Uppfyller inte »delbara med 100 men inte delbara med 400».

```
~(mod(y, 100)==0
& mod(y, 400)~=0): 1    1    1    1    1    0    0    0    1    0
leapgreg(y):       1    1    0    1    1    0    0    0    1    0
```



# Triangelanalys

Avgör om en triangel med sidorna `a`, `b` och `c` är liksidig, likbent, oliksidig, eller ickeexisterande.

```
% Analyserar triangeln med sidorna a, b och c.
if a<=0 | b<=0 | c<=0 'ickeexisterande'
elseif a>=b+c | b>=c+a | c>=a+b 'ickeexisterande'
elseif a==b & b==c & c==a 'liksidig'
elseif a==b | b==c | c==a 'likbent'
else 'oliksidig'
end
```

En text/sträng är en radvektor med element av typ tecken/char. Det som lagras är teckenkoderna enligt ISO 8859-1, Latin1.

```
uint16('aåäöAÅÄÖ -')
ans = [97 229 228 246 65 197 196 214 32 45]
char(65)
ans = A
```



# Triangelanalys forts.

Om vi lägger kommandona i filen tri.m så kan vi använda den som kommandofil.

```
a = 0; b = 1; c = 2; tri % 0 1 2
ans = ickeexisterande
a = 3; tri               % 3 1 2
ans = ickeexisterande
a = 1.5; tri             % 1.5 1 2
ans = oliksidig
a = 2; tri               % 2 1 2
ans = likbent
b = 2; tri               % 2 2 2
ans = liksidig
```



# Avlusning (eng. debugging)

För att följa ett programs exekvering kan man:

- Göra kontrollutskrifter på valda platser
- Sätta in »brytpunkter» där exekveringen tillfälligt stoppas
- Sätta in »villkorliga brytpunkter» där exekveringen tillfälligt stoppas om ett villkor är sant
- »Stega» i programmet
- Indela programmet i »celler» och exekvera en cell i taget
- Använd verktyget `mlint` för att hitta förslag till förbättringar

# Switch-kommando

När man vill exekvera olika kommandon för olika värden på ett uttryck kan man använda `switch` i.s.f. `if`.

`switch` *uttryck*
`case` värde1
  kommandon1
`case` värde2
  kommandon2
…
`otherwise`
  kommandon
`end`

Värdena är uttryck eller följder av värden inom »{}» (klammerparenteser) dvs. en celltabell. Om uttryckets värde inte finns med bland värdena utförs kommandona efter `otherwise`. Delen `otherwise` *kommandon* får utelämnas.



# Switch-kommando forts.

Låt `t` vara resultatet av ett tärningskast:

```
switch t
case {1,3,5}
  'Udda antal prickar'
case {2,4,6}
  'Jämnt antal prickar'
otherwise
  'Tärningen verkar onormal!'
end
```



# Felhantering, try-kommandot

Vissa typer av fel får Matlab att skriva felmeddelande.

```
a = [1]; a(3)
??? Index exceeds matrix dimensions.
```

Man kan fånga upp felet och skriva ett eget felmeddelande eller försöka rätta felet.

```
a = [1];
try
  a(3)
catch
  '??? Indexfel'
end
??? Indexfel
```



# Felhantering forts.

Man kan själv rapportera fel med `error(`*felmeddelande*`)`. Då skriver Matlab var felet inträffade och felmeddelandet. Om ett sådant fel rapporteras i en m-fil som anropas mellan `try` och `catch` skrivs inte felmeddelandet, utan kommandona mellan `catch` och `end` utförs.

Vissa beräkningar ger inte felmeddelande men möjligen varningsmeddelande och ett speciellt felvärde som resultat. `3/0` och `2*realmax` ger båda resultatet `Inf` (∞) och 0/0 ger `NaN` (not a number). När man konverterar ett värde till någon datatyp som inte innehåller värdet lagras det närmaste möjliga värdet.

```
int8(1000)
ans = 127
```

# Repetitionskommando, slinga, for

Att utföra samma kommandon många gånger är vanligt. När man upprepar för en följd i förväg kända värden använder man `for` och när man upprepar så länge ett villkor är sant använder man `while`.

```
for variabel = uttryck
   kommandon
end
```

Variabeln kallas slingans styrande variabel. Den sätts i tur och ordning till kolumnerna i den tabell som är resultatet av uttrycket. Uttrycket beräknas bara när exekveringen av **for**-slingan börjar. Uttrycket är ofta en linjär talföljd, `b:s:e`.



# Kuggväxel

Man vill konstruera en kuggväxel där utväxlingen kommer så nära ett givet värde, *gear*, som möjligt. Man har tillgång till kugghjul med minst *min* och högst *max* kuggar. Läs in *gear*, *min* och *max*.

- Läs *min*, *max* och *gear*.
- Kontrollera att *min*, *max* och *gear* är större än noll.
- Låt *m* gå från *min* till *max*.
- Testa de två *n* som ligger på var sin sida om *m*/*gear*.
- Håll reda på de bästa *m* och *n* hittills dvs. de som givit det minsta värdet på abs(*m*/*n*-*gear*).
- Skriv ut bästa värdena.



```
min = input('Minsta kuggantal: ')
max = input('Största kuggantal: ')
gear = input('Önskad utväxling: ')
bestdiff = realmax;
if gear>0 & max>0 & min>0
  for m = min:max
    nprel = floor(m/gear);
    for n = nprel:nprel+1
      if n<min n=min; end; if n>max n=max; end;
      diff = abs(m/n-gear);
      if diff<bestdiff, bestdiff = diff;
         bestm = m; bestn = n; end % if
    end % for n
  end % for m
  disp(['Bästa m, n och m/n är ' int2str(bestm)...
  ', ' int2str(bestn)' och ' ...
  num2str(bestm/bestn)]); end % if
```



# Kuggväxel forts.

Antag att kommandona finns i kommandofilen vaxel.m.

```
vaxel
Minsta kuggantal: 5
Största kuggantal: 50
Önskad utväxling: 3.14159
Bästa m, n och m/n är 22, 7 och 3.1429
```

Funktionen `disp(`*sträng*`)` skriver ut sin parameter utan omgivande strängparenteser (`'`). I kommandofilen kan vi med samma resultat skriva:

```
['Bästa m, n och m/n är ' int2str(bestm) ...
', ' int2str(bestn) ' och ' ...
num2str(bestm/bestn)]
```

# While

`while` *villkor*
  kommandon
`end`

Villkoret beräknas när exekveringen av **while**-slingan börjar och efter varje varv. Exekveringen avbryts när villkoret är falskt. Slingor och villkorskommandon kan innehålla andra slingor och villkorskommandon.

Kommandot `break` innebär att närmast omgivande slinga avbryts och exekveringen fortsätter efter slingans avslutande `end`. Kommandot `continue` innebär att pågående varv i närmast omgivande slinga avbryts och nästa varv, om något återstår, exekveras.



# eps

Ett mått på en numerisk datatyps precision får man med det minsta tal, *systeps*, för vilket 1+*systeps*≠1.

```
systeps = 1;
for i = 1:1000
  systeps = systeps/2;
  if systeps+1==1
    break
  end
end
systeps = systeps*2
systeps = 2.2204e-16
```

I Matlab finns en inbyggd variabel `eps` med motsvarande betydelse. Matlabs `eps` kan ändras och påverkar den relativa feltoleransen för vissa kommandon.

```
eps = 2.2204e-16
```



# Testning – är ett program korrekt?

Att ta reda på om ett program är korrekt är svårt.

- Hur beskriver (specificerar) man vad programmet ska göra? Hur tolkar man uppdragsgivarens eller framtida kunders önskemål?
- Man kan systematiskt testa olika fall/indata. Det är sällan möjligt att pröva alla fall/indatakombinationer. När man hittar ett fel kan man försöka rätta detta.
- Man kan systematiskt resonera om ett program och jämföra resonemangets resultat med programmets specifikation.
- En *algoritm* (av Abu Ja'far Mohammaed ibn Mûsâ al-Khowârizmî som 825 gav ut en algebrabok *Kitab al jabr w'al-muqabala*) är en detaljerad beskrivning av hur man i ett ändligt antal steg löser ett problem.



# Filläsning- och skrivning

Läs tal från filen in.dat tills talet 0 hittas eller filen är slut och skriv de positiva talen på filen out.dat.

Man läser alla värden från en fil till en variabel med `load`. Vi vill läsa ett värde i taget. Vi öppnar filen med `fopen` och om det går bra får vi en filreferens. Om det inte går bra returneras –1. Därefter kan vi läsa med `fscanf` och skriva med `fprintf`.

*filreferens* `= fopen(`filnamn,behörighet`)`
returnerar en filreferens, behörigheten utelämnas eller skrivs `'r'` för läsning, skrivs `'w'` för skrivning från början och skrivs `'a'` för skrivning efter tidigare filslut (append).

# Filläsning- och skrivning forts.

```
frin = fopen('in.dat');
frout = fopen('out.dat','w');
% Testa helst att fopen ej har givit -1
[x,count] = fscanf(frin,'%d',1);
if count>0
  while x~=0
    if x>0
      fprintf(frout,'%d\n',x);
    end
    [x,count] = fscanf(frin,'%d',1);
    if count==0 break, end
  end % while
end % if
fclose(frin);
fclose(frout);
```



# Filläsning och -skrivning forts.

Vi lägger koden i kommandofilen io.m och testar.

```
type in.dat
2 3 5 -7 11
13 -17 0 19 -23 29
io
type out.dat
2
3
5
11
13
```

Och med ändrad infil.

```
type in.dat
2 3 5 -7 11
13 -17 19 -23 29
io
type out.dat
2
3
5
11
13
19
29
```

# Filläsning- och skrivning forts.

`[`*var*`,`*läst_antal*`] = fscanf(`*filreferens*`,`*formatsträng*`,`*önskat_antal*`)`
returnerar en variabel med värden från filen i *var* och antalet lästa element i *läst_antal* (*läst_antal* får utelämnas), *formatsträng* beskriver hur data i filen ska tolkas (`%d` betyder heltal) och *önskat_antal* anger hur många värden som önskas.

`fprintf(`*filreferens*`,`*formatsträng*`,`*val1*`, …)`
skriver värdena *val1*, … på filen enligt *formatsträng* (`%d\n` betyder som heltal med radbyte efter varje tal).