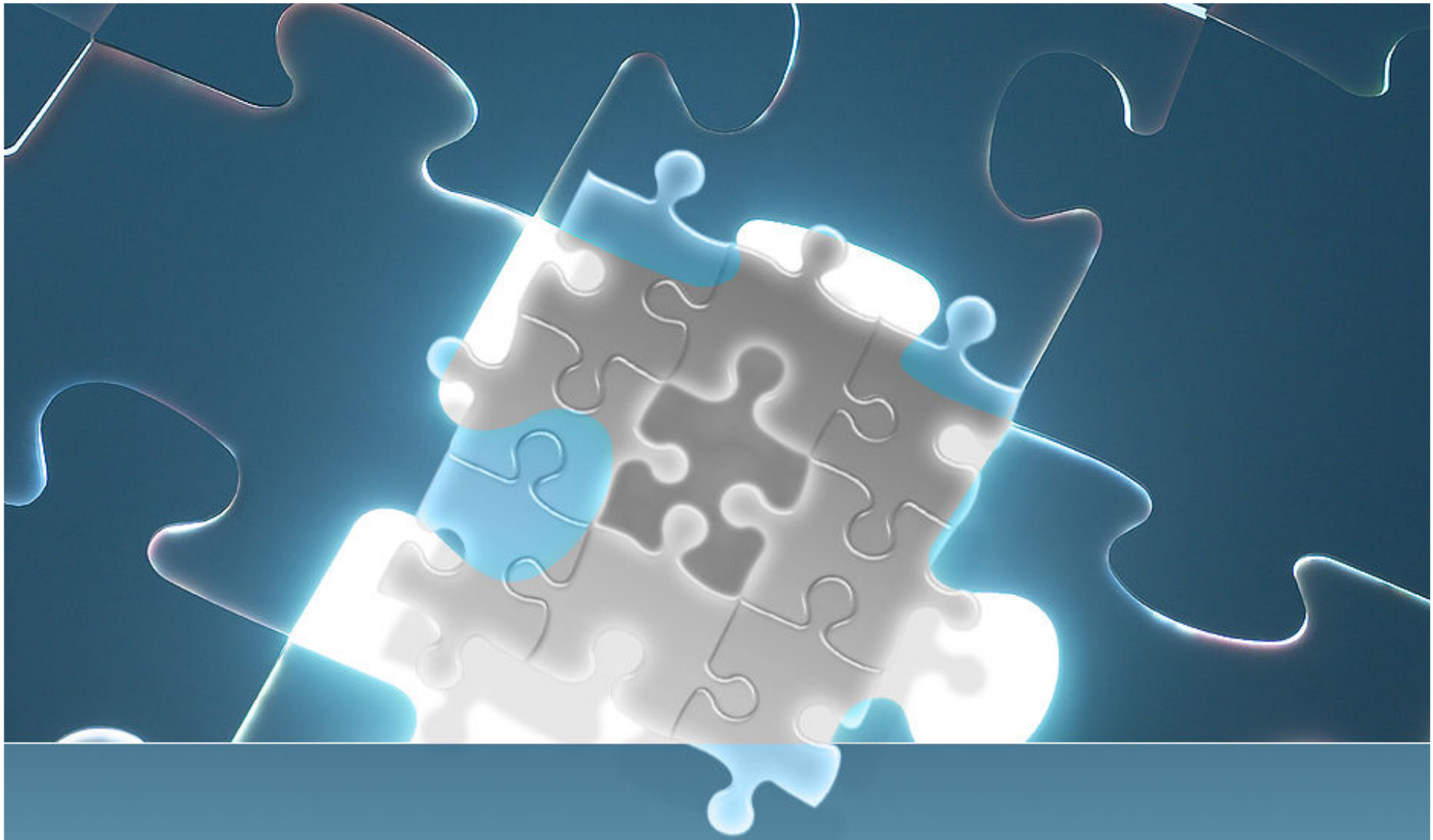


Algoritmer och interaktiv Python

Linda Mannila 11.9.2007

Denna föreläsning

- Räkneövningstider
- Algoritmer
- Interaktiv Python
  - Datatyper
  - Variabler
  - Typning

Repetition

- Vad vi än skall göra måste vi veta hur vi skall bära oss åt för att få det gjort
  - Kan inte köra bil om vi inte vet hur man gör
  - Omöjligt (åtminstone himla dumt ☺) att hoppa i 10 m djupt vatten om vi inte kan simma
  - MEN - vi kan gå i bilskolan och ta simlektioner för att lära oss
- En dator *vet inte hur den skall göra något* och *kan inte lära sig*
  - Användaren måste förklara vad datorn skall göra och hur → *programmering*

Repetition

- För att få en dator att göra något måste vi
  - skriva ett program som exakt berättar för datorn *vad* den skall göra, och
  - *hur* den skall göra det, steg för steg
- Datorn kan sedan *exekvera* (köra) programmet
  - utför varje steg tills det slutliga målet är nått
- *Instruktioner* (satser) är den verkliga grunden i ett program, och används för att manipulera data, utföra beräkningar, skriva ut data på skärmen osv.

Program skapas i faser

1. Definiera problemet
2. Identifiera input och output
3. Designa en algoritm
4. Representera algoritmen (flödesschema och/eller pseudokod)
5. Skriv programkoden
6. Testa programmet (kör det)
7. Debugga programmet (fixa eventuella fel)

Vad är en algoritm?

- En mängd steg-för-steg instruktioner som berättar exakt hur ett problem skall lösas (ett "recept")
- Exakta, entydiga och fullständiga instruktioner
- Problemet måste vara välspecifierat
- Algoritmer kan uttryckas
  - med *vanligt språk* (svenska, finska, kinesiska),
  - i ett *programmeringsspråk*,
  - med *pseudokod* eller
  - med *flödesscheman*

Vad är ett program?

- En algoritm som *implementerats* så att den kan utföras av en dator
- Uttrycks i ett programmeringsspråk

Fasters resa

Problem

Din gamla faster från Amerika skall komma på besök, men du hinner inte möta henne på flygfältet. I stället måste du henne instruktioner för hur hon kan ta sig till ditt hus på egen hand. Du ger henne tre olika alternativ:

Ta en taxi

1. Gå till taxi-stationen
2. Hoppa in i en taxi
3. Ge chauffören hemadressen

Hyr en bil

1. Gå till biluthyrningsdisken på flygfältet
2. Hyr en bil
3. Använd kartan för att hitta ända fram

Ta bussen

1. Ta buss 57 till centrum
2. Byt till buss 61
3. Stig av på Rådhusgatan
4. Gå två kvarter söderut

Fasters resa

- Alla alternativ är en *algoritm*, dvs. en samling steg-gör-steg-instruktioner
- Alla tre algoritmer leder till samma resultat, men på olika sätt
  - Även inom programmering flera olika algoritmer för hur man kan utföra en viss uppgift eller lösa ett problem
- Varje algoritm har sina för- respektive nackdelar
  - Taxi – snabbt men dyrt
  - Bil – kräver mer av din faster men flexibelt
  - Buss – långsamt men billigt
- Man väljer algoritm beroende på omständigheterna

En algoritm måste...

- *vara välordnad*
  - klart i vilken ordning de olika instruktionerna skall utföras
- *innehålla entydiga instruktioner*
  - inte kunna tolkas på olika sätt, ingen tvekan om betydelsen
- *kunna utföras*
  - alla instruktioner som ingår måste gå att utföra. Ej t.ex. instruktion som kräver att vi dividerar med noll
- *alltid ge ett resultat*
  - annars omöjligt veta om algoritmen har utförts eller om den ännu är på hälft
- *avslutas inom ändlig tid*
  - inga evighetsprogram, måste alltid komma till ett slut

Koka te

Version 1:

1. Koka vatten
2. Sätt tepåsen i en kopp
3. Häll vatten i koppen

Koka te

Version 2:

1. Fyll en kastrull med vatten.
2. Sätt kastrullen på spisplattan.
3. Vänta tills vattnet kokar.
4. Ta av kastrullen från plattan.
5. Ta fram en tepåse.
6. Sätt påsen i en kopp.
7. Lyft vattenkastrullen.
8. Luta kastrullen så att vattnet rinner ner i koppen.
9. Vänta tills koppen är full.
10. Ställ tillbaka kastrullen på plattan.

Poängen?

- När man programmerar måste man vara

  otroligt exakt och detaljerad

- Precis *allt* som datorn skall göra måste skrivas ut

Att skapa program

- Ett program är helt enkelt en lösning på ett problem
  - Det svåra ligger i att komma på lösningen
- För att ett program skall fungera som man tänkt: *för- och eftervillkor*

  Förvillkor => Program => Eftervillkor

- *Om* förvillkoren är uppfyllda då man startar programmet *kommer* eftervillkoren att vara uppfyllda då man har kört programmet

Att skapa program

- *”Jag är snål på sockerkaka men har ingen. Hur skall jag få en?”*

- Vissa förutsättningar som måste gälla för att du skall kunna lösa problemet
  - ingredienser som behövs
  - en funktionsduglig ugn
  - kakform
  - veta hur man bakar en kaka, dvs. ett recept

  → Kakbakningens *förvillkor*

Att skapa program

- **Förvillkor:** Du har alla ingredienser och annat som behövs
- **Program:** Receptet
- **Eftervillkor:** Nygräddad sockerkaka

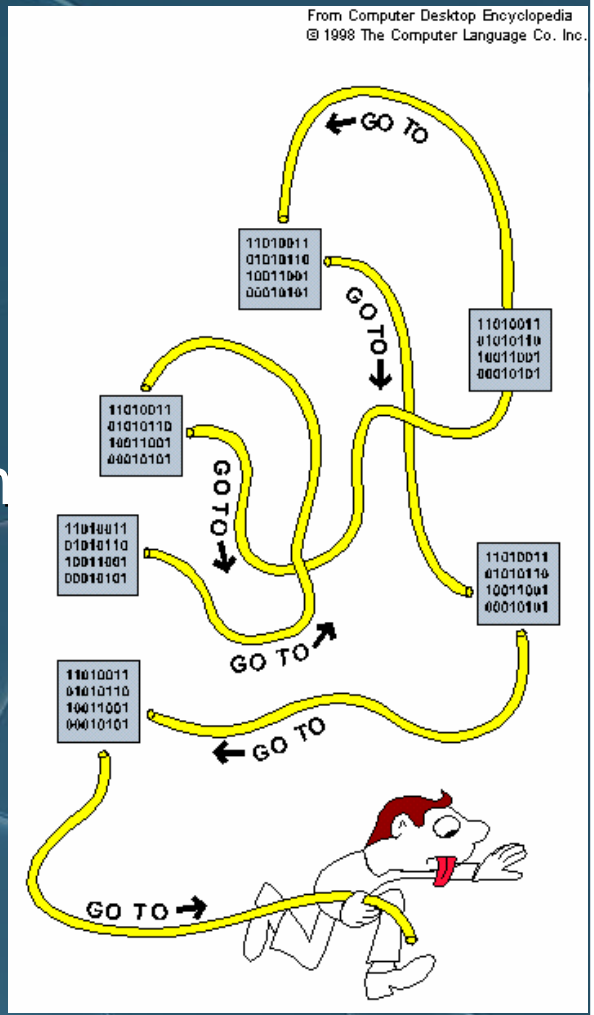
Att skapa program

- **Förvillkor:** Det som måste gälla för att programmet skall kunna köras
- **Program:** Den implementerade algoritmen / koden
- **Eftervillkor:** Det resultat man är ute efter

Spaghettikod

- Första programmen skrivna i högnivåspråk var ofta svåra att läsa och förstå
  - användningen av GOTO-satser, (berättar för datorn att hoppa till en viss rad i programmet)

  → spaghettikod, massa hopp fram och tillbaka i koden, svårhittade fel



Spaghettikod, exempel

```
10 i = 0
20 i = i + 1
30 print i; " squared = "; i * i
40 if i < 10 then goto 20
50 print "Program Completed."
60 end
```

- Problem?

Strukturerad programmering

- Edsger Dijkstra (1930-2002)
- Mål: program som är lättare att skriva, läsa och underhålla
- Samma utgångspunkt: en mängd instruktioner som tillsammans bildar ett program.
- I stället för hoppsatser:
  - Tre olika typer av instruktioner
    - Vissa skall utföras en gång
    - Upprepas
    - Kanske inte utföras alls

Tre instruktionstyper

Sekvens: allt i tur och ordning

Först – sedan – sist

Villkorliga: valmöjlighet, kanske inte utförs alls

Om... så...

Iterativa: repeteras, om och om igen

Upprepa, tills, medan, ...

No more spaghetticode

```
10 i = 0
20 i = i + 1
30 print i; " squared = "; i * i
40 if i < 10 then goto 20
50 print "Program Completed."
60 end
```

- Strukturerad, anti-spaghettiversion

```
for i in range(1,10):
    print i, "squared =", i ** 2
print "Program Completed."
```

Representera algoritmer

- Algoritmer → planera program med papper och penna, utan dator och programmeringsspråk
  - lösningsmodell som sedan kan implementeras och exekveras
- Hittills uttryckt i naturligt språk
- Mer exakta alternativ:
  - pseudokod
  - flödesscheman

Pseudokod & flödesscheman

- *inte* programmeringsspråk
- låter oss strukturera algoritmer (program) klart och tydligt *utan* att behöva använda ett *programmeringsspråk*
- viktiga när man designar program

Pseudokod vs. flödesscheman

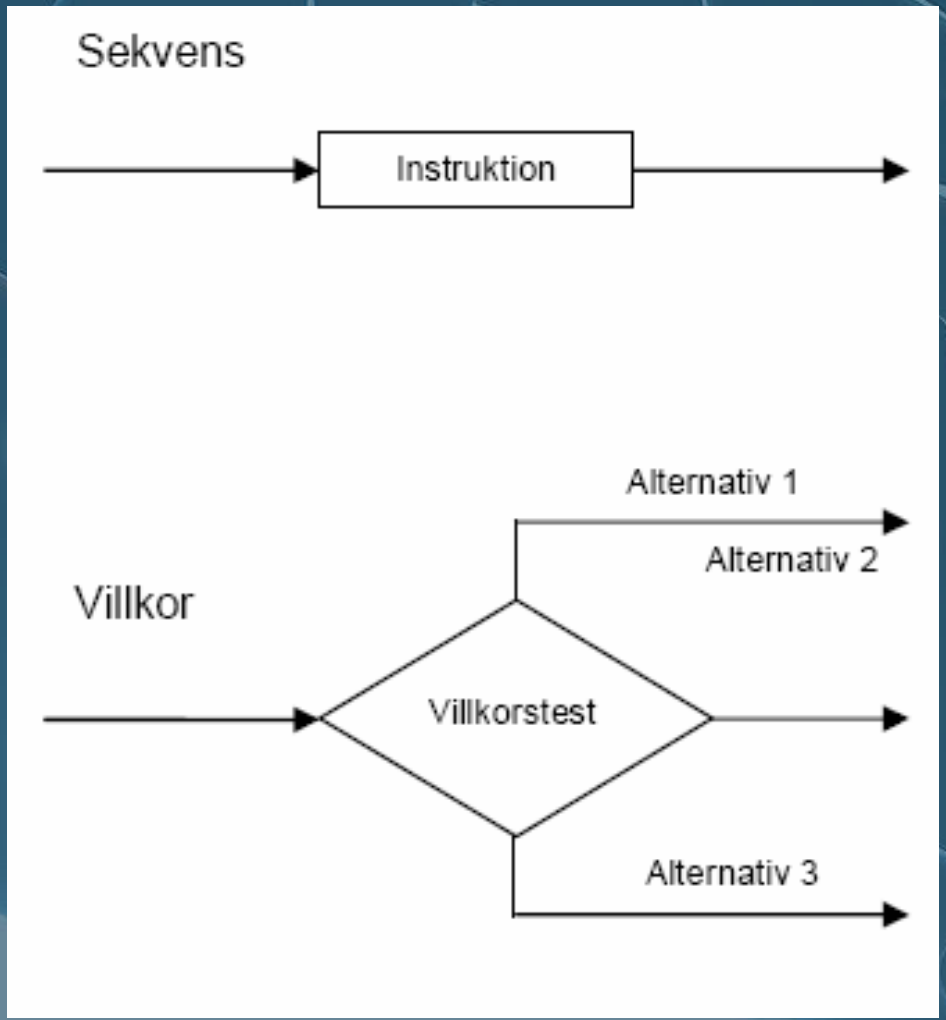
Pseudokod (”nästan kod”)

beskriver algoritmer exakt med ord

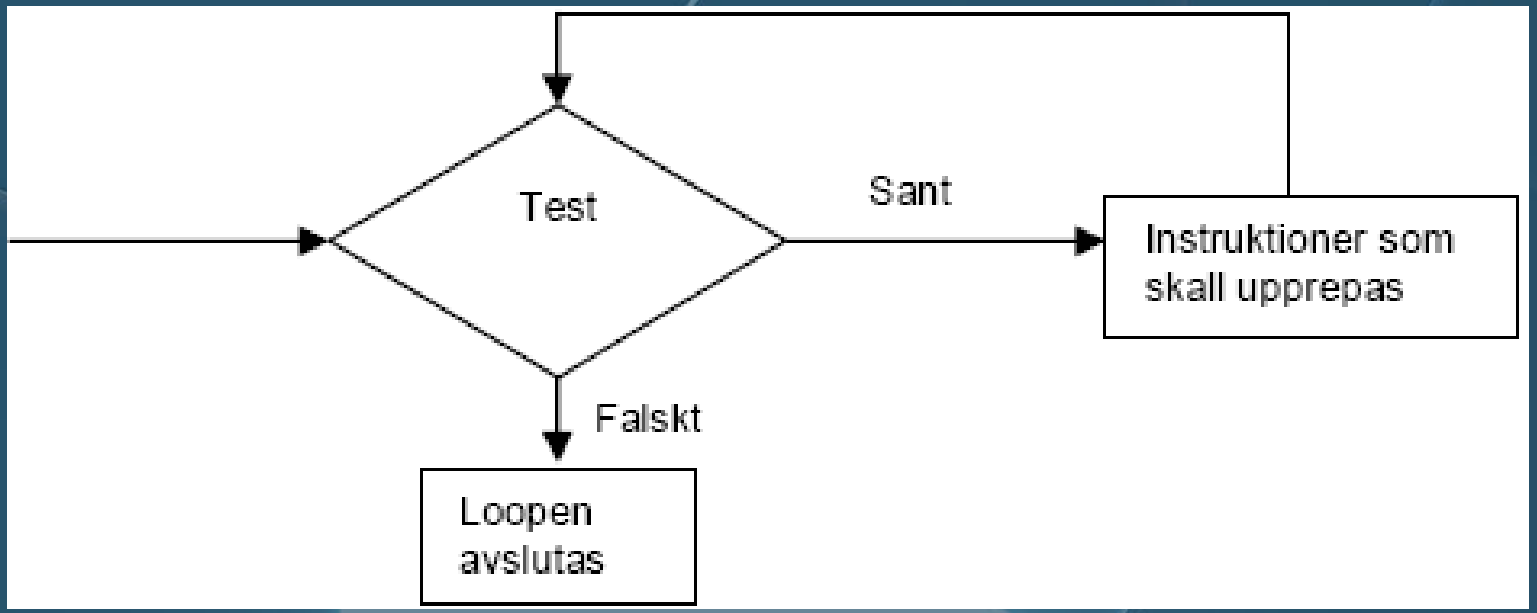
Flödesscheman

beskriver algoritmer exakt grafiskt med figurer

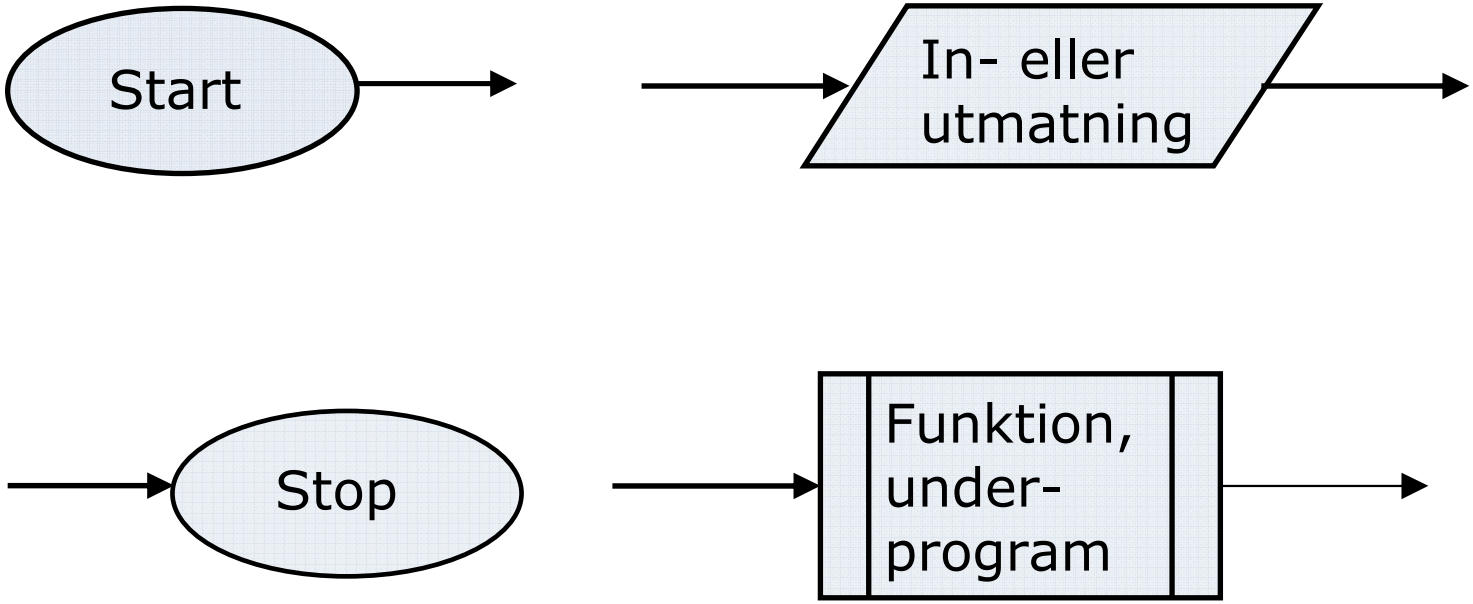
Flödesscheman, symboler



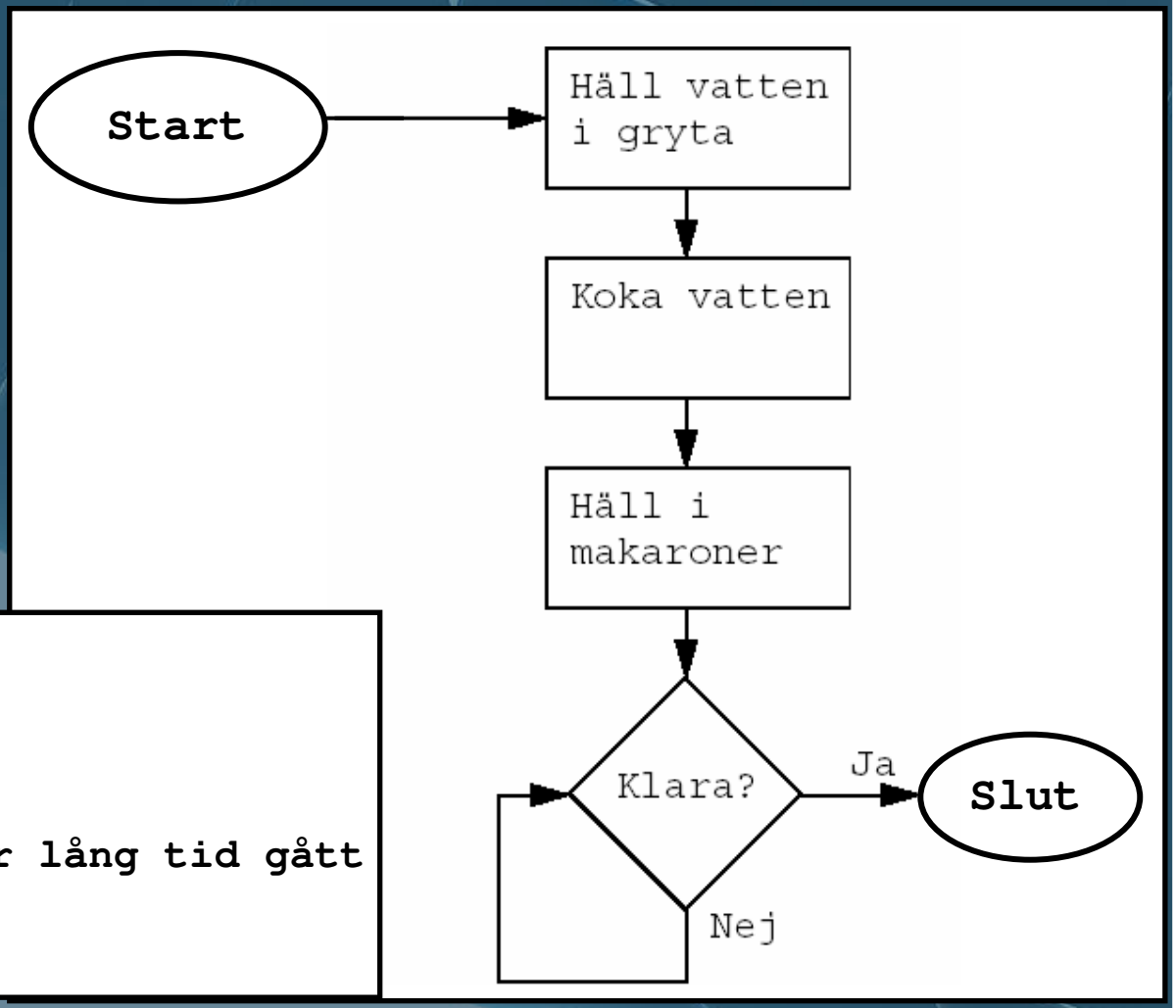
Flödesscheman, symboler



Flödesscheman, symboler



Koka makaroner




```
vatten → gryta
koka vatten
makaroner -> gryta
tills makaroner mjuka eller lång tid gått
    låt makaroner koka
```

Algoritmer i algoritmer

Vanlig approach:

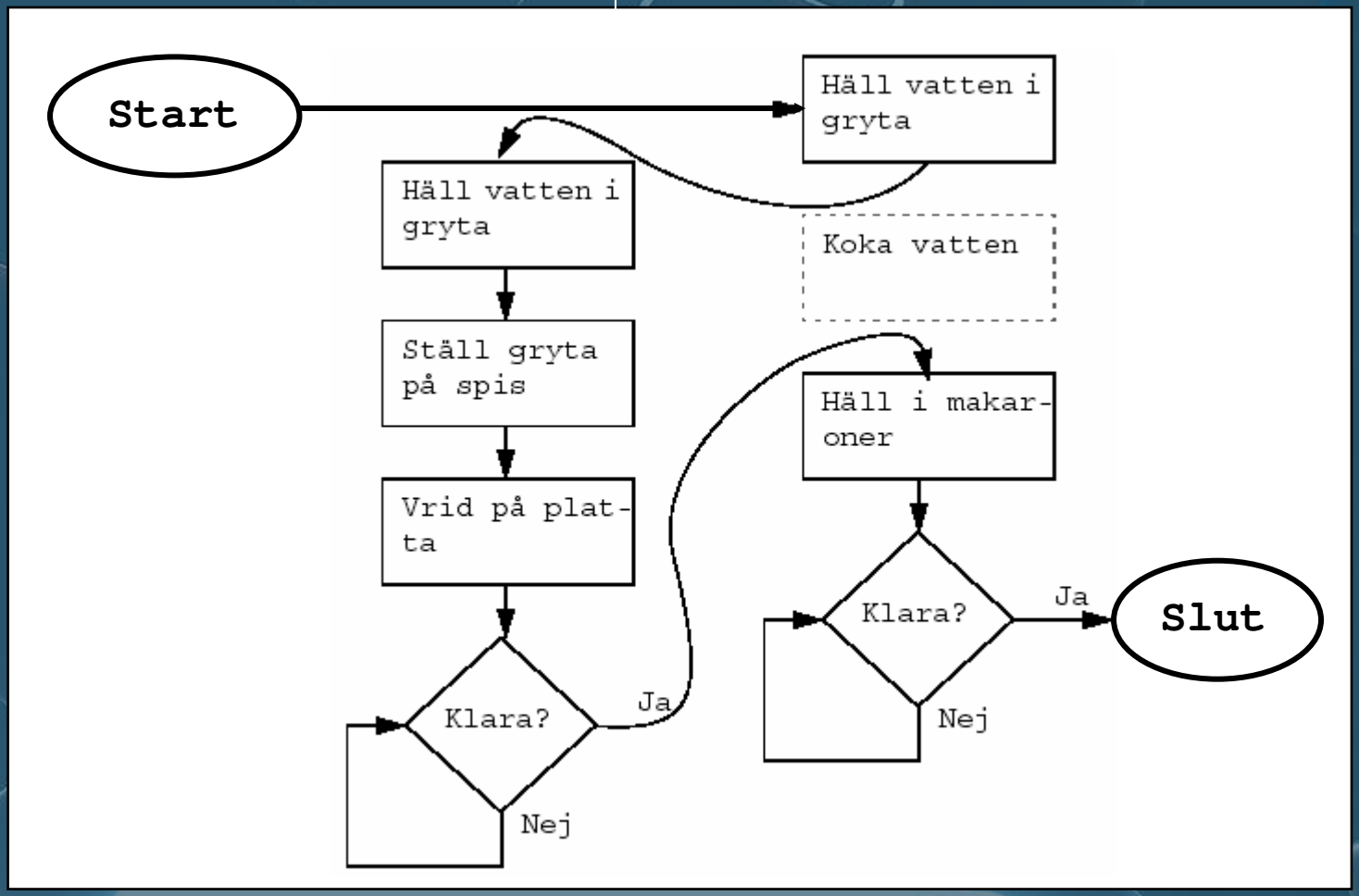
Dela problemet i mindre delar (*”divide and conquer”*)

Här t.ex. egen algoritm för att *koka vatten*

Input: vatten, gryta, spis

1. Häll vatten i gryta
2. Ställ gryta på spis
3. Vrid på platta
4. Vänta tills vatten kokar

Algoritmer i algoritmer



Uppgift

- Parvis – minst en i paret bör ha snörade skor
- Skriv en algoritm för hur man knyter skosnören
- Pseudokod, flödesschema eller båda

Äntligen...

Dags att börja programmera

- Interaktiv Python -

Repetition

- Python är ett tolkat språk
- Den interaktiva prompten
  - >>>
  - Instruktionerna körs (exekveras) genast de skrivs in
  - Omedelbar feedback – du ser resultatet genast

Python som räknemaskin

- Skriv in beräkningar vid prompten och tryck enter

- Vanliga aritmetiska operationer:

Operation	Symbol	Exempel
Addition	+	`1 + 2 == 3`
Subtraktion	-	`4 - 3 == 1`
Multiplikation	*	`2 * 3 == 6`
Division	/	`14 / 3 == 4`
Rest	%	`14 % 3 == 2`
Upphöjning	**	`5 ** 2 == 25`

Python som räknemaskin

- Operationerna utförs i samma ordning som i matten:
  1. parenteser ()
  2. exponenter **
  3. multiplikation *, division /, och rest %
  4. addition + och subtraktion -

Datatyper

- Värden kan vara av olika typ
  - t.ex. text är inte ett tal → behöver en egen typ
- Primitiva datatyper (*eng.* primitives)
  - Grundläggande typer, bastyper
  - Motsvarar ett enda värde
- Samlingar (*eng.* collections)
  - Kombinationer av de primitiva
  - Kan innehålla flera värden

Datatyper: Tal

- Heltal (*eng.* integer)
  - tal *utan* decimaldel
  - 13, 23, 1128, -99 etc.
- Flyttal (*eng.* Float)
  - tal med decimaldel
  - 2.3, -0.0076, 19.89, 6.0 etc.
  - Obs! Punkt, inte komma!
- Komplexa tal (*eng.* complex)
  - 4+3j
- Long
  - Stora heltal
  - 62081372132

Division

- När ett heltal divideras med ett annat utförs heltalsdivision:
  - 12 / 5 = 2
  - 1 / 3 = 0
- För att få reda på resten kan man använda rest-operatorn (%)
  - 12 % 5 = 2
  - 1 % 3 = 1

Flyttalsdivision

```
>>> 12 / 5
2

>>> 12.0 / 5
2.3999999999999999

>>> 12 / 5.0
2.3999999999999999

>>> 12.0 / 5.0
2.3999999999999999
```

För att få flyttalsdivision måste *minst ett av talen vara ett flyttal*.

Datatyper: Strängar

- Text i programmeringsspråk kallas för *strängar*
- Sträng = text innanför citationstecken
- Både ' och " kan användas

```
'Detta är en sträng'
"Och det här"
```

- Strängar kan innehålla citationstecken

```
'Han sade: "Hoppsan" högt'
```

Utskrift på skärmen

- Python

```
>>> print "Hallå där världen!"
Hallå där världen!
```

- Java

```
class Hello {

        public static void main (String [] args) {
                System.out.println("Hallå där världen!");
        }
}

% javac Hello.java
% java Hello
Hallå där världen!
```

Strängoperationer

- Både + och * kan användas med strängar
  - + operatorn *konkatenerar* (fogar ihop) två strängar

```
>>> print “Hej" + “då!”
```

    skriver ut *Hejdå!*

  - Operatorn * repeterar en sträng

```
>>> print 3 * "ha"
```

    skriver ut *hahaha*

Bart Simpson

```
>>> print "I won't teach others to fly" * 100
```

Skriver ut texten 100 gånger!

Men, efter varandra
– hur få dem på skilda rader?

Bart Simpson

```
>>> print ("I won't teach others to fly" + "\n") * 100
```

eller

```
>>> print ("I won't teach others to fly\n" * 100
```

Specialtecknet "\n" skriver ut en ny rad.

Foga ihop strängar med tal

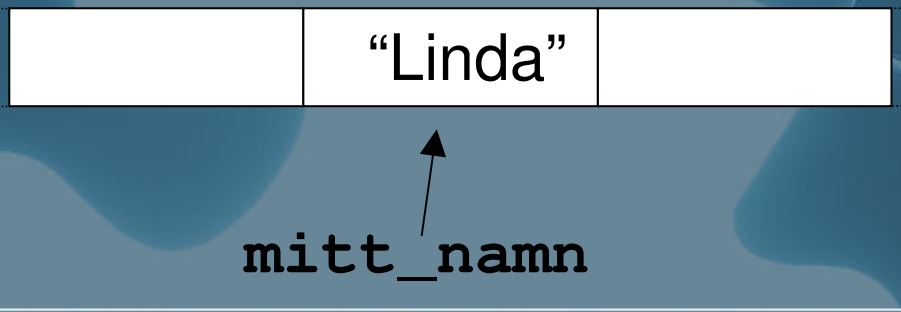
- Använd *kommatecken* för att foga ihop strängar med t.ex. tal

```
>>> print "2 + 2 blir", 2 + 2

>>> print "3 * 4 blir", 3 * 4

>>> print 100 - 1, "är det samma som 100 - 1"
```

Variabler

- Variabel = namn som refererar till ett värde
- Plats i datorns minne för att lagra data så att vi kan komma åt det senare igen.

- Variabeln är ett namn som refererar till ett värde (pekar på den minnescell som innehåller det värdet)



Variabler

- I Python skapas en ny variabel samtidigt som man ger den ett värde. Detta görs genom en och samma tilldelningssats

```
>>> mitt_namn = 'Linda'
```

- = kallas *tilldelningsoperator*

- Efter tilldelningen kan man använda variabelnamnet var som helst i koden där ett värde av samma datatyp passar in

```
>>> print 'Hej', mitt_namn
Hej Linda
```

Variabler

- **OBS!** En variabel som lagrar en primitiv datatyp kan bara innehålla ett värde i taget! Om man tilldelar en variabel ett nytt värde skrivs det gamla över!

```
>>> mitt_namn = 'Linda'
>>> print mitt_namn
Linda
>>> mitt_namn = 'Linus'
>>> print mitt_namn
Linus

>>> age = 45
>>> age = age +1
>>> print age
46
```

Variabelnamn

- Variabelnamn får ***inte***
  - vara ett nyckelord
  - börja med en siffra
  - innehålla otillåtna tecken (t.ex. $, %, ?, =)
- Skillnad på små och stora bokstäver
  - `minBil` är inte samma variabel som `minbil`
- Vänj dig från början vid att använda namn som betyder något:
  - Dåligt: `vd = 7`
  - Bra: `veckodagar = 7`

Typning

- Python har dynamisk typning
  - kan själv avgöra vilken datatyp en variabel har
  - `type`(*variabelnamn*)
- Andra programmeringsspråk har ofta statisk typning
  - programmeraren måste deklarera alla variabler till den datatyp som den skall kunna innehåll

Typning

- Python:

  `vikt = 55.5` (Python vet att vikt är en flyttalsvariabel)

  `age = 25` (Python vet att age är en heltalsvariabel)

  `ord = 'dag'` (Python vet att namn är en strängvariabel)

- Statiskt typade språk (t.ex. Java):

```
float vikt = 55.5
int age = 25
String ord = 'dag'
```