Mer programmering för alla!

Fördjupning för lärare i grundskola och gymnasium

Björn Regnell

Professor
Datavetenskap, LTH, Lunds universitet

lth.se/programmera

2014-10-24

Björn Regnell, Lunds universitet

Programmering för elever

Foto Charlotte Carlberg Bärg

Programmering för lärare

Foto: Håkan Elofsson

Verktyg för engagemang

– Jag har jobbat som lärare i 14 år, men det är först på mina kodlektioner som jag känner att undervisningen är som den *borde* vara. Eleverna löser problem på riktigt och är intresserade av det!

Karin Nygårds, svensklärare på mellanstadiet

Publicerat 2013-12-16 på *Lärarnas Nyheter*
http://www.lararnasnyheter.se/alfa/2013/12/16/hittar-koden-tekniken

Programming UNPLUGGED

Hitta på ditt eget dansprogrammeringsspråk!

http://teacherhack.com/programmering/dansprogrammering/

Lärandemål fördjupning

1. Kännedom om några grundläggande principer i datalogiskt tänkande
   - sekvens, repetition, abstraktion, alternativ, variabel, objekt, klass, ...
2. Exempel på progression i datalogiskt tänkande
3. Kännedom om några lärandetrösklar
4. Praktiska programmeringstips

Exempel på "eviga" färdigheter som utvecklas med datalogi och programmering

- Tänka logiskt
- Tänka steg-för-steg
- Tänka i abstraktioner
- Konstruera abstraktioner
- Skriva exakt – allt du skriver har en effekt
- Fantasi och kreativitet
- Förklara för varandra
- Samarbete
- ...

"Computational thinking"
Datalogiskt tänkande

Diskussion om programmeringslärande

- Hur lär man sig att programmera?
  - Skapa mental modell av vad som händer genom konstruktion
  - Prova själva, steg för steg
  - Förmåga att hitta kompileringsfel och buggar genom att prova sig fram
  - Hjälp över trösklar och samtidigt utveckla abstrakt tänkande
  - Abstraktionsmekanismerna som kommer till din räddning:

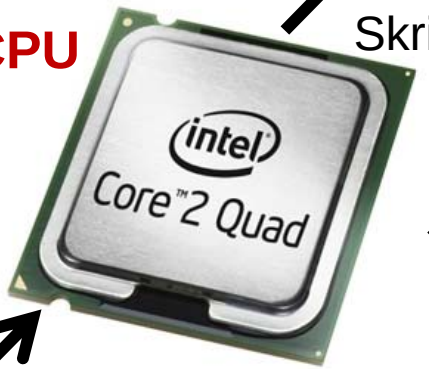
    "**tack för upprepa & def - nu kan jag programmera det jag vill åstadkomma mycket enklare!**"
  - Olika **lärandestilar** är mer eller mindre lämpliga för olika individer.
    Se till att undervisningen ger möjlighet till flera olika lärandestilar:
    att **se**, att **höra, diskutera, motorisk**, att göra, rita, skriva för hand
  - De som **reflekterar** över sin egen lärandestil lär sig mer!
    Vad har jag lärt mig nu? Varför är detta bra att kunna? Hur lär jag mig bäst? ...
  - Vad skiljer vuxnas lärande jämfört med barns?

Hur funkar en dator?

Utdata-enhet
data till omvärlden

Skriv ut

Centralprocessorenhet CPU
bearbetar *data* enligt ett *program*



Läs & Skriv

Minne
sparar både
data och *program*

Läs in

Indata-enhet
data från omvärlden

Uppdelning: data & instruktioner

- **Heltal** används för att representera all **data**:
  bokstäver, bilder, musik, filmer, etc.
- **Heltal** används för att representera **instruktioner**
  (program) som anger vad datorn ska göra:
  addera, multiplicera, skriva i minnet, etc.
- I datorns minne finns **både** data **och** instruktioner
- En **kompilator** hjälper till att översätta (för människor)
  mer lättbegripliga instruktioner till rätt heltal

Källkod

(lättare för människor)

Maskinkod

(siffror som styr datorn)

Scala och Java

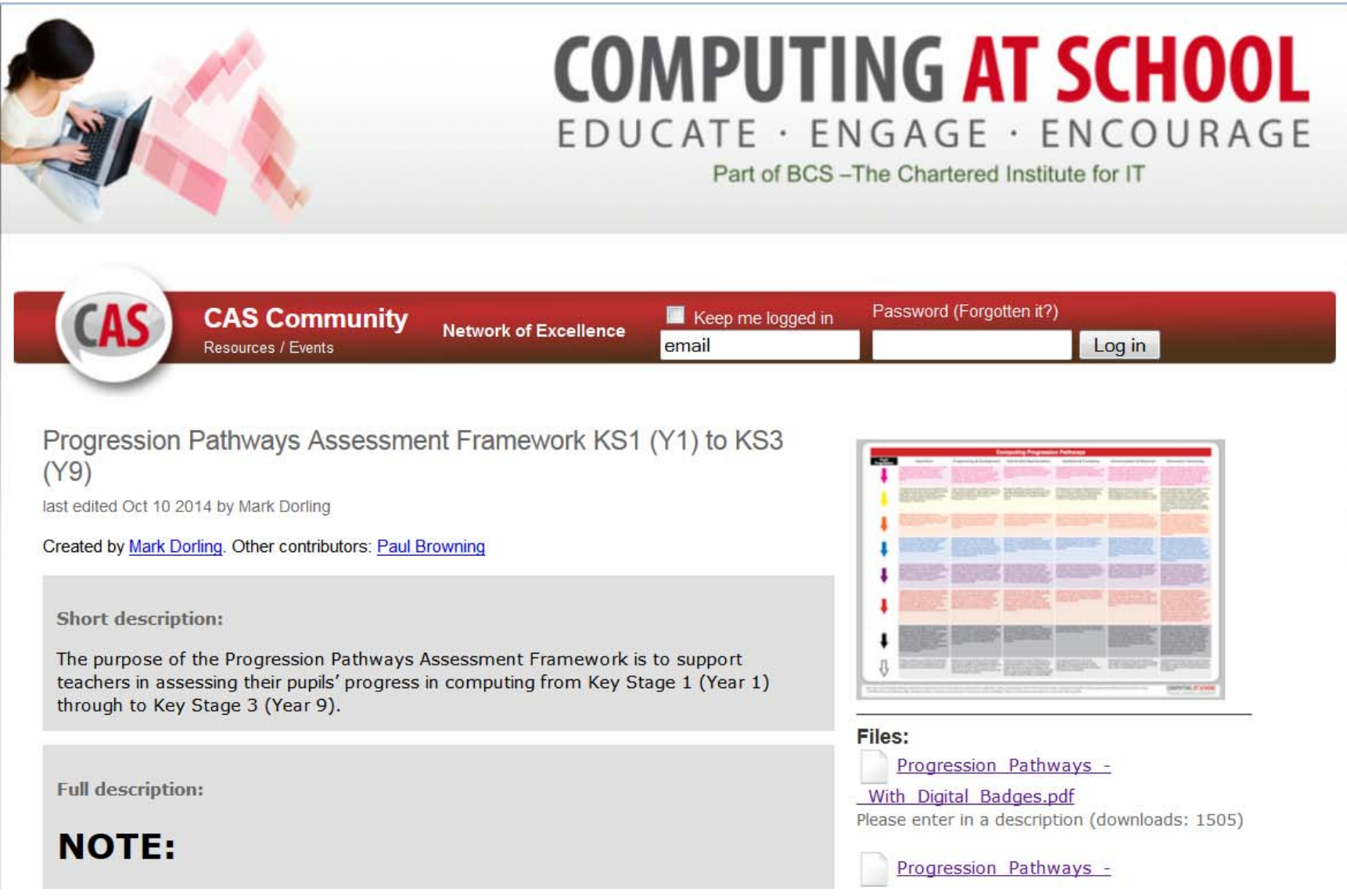
- Scala är ett modernt och kraftfullt programspråk som bygger vidare på programspråket Java
- Java är det mest använda programspråket (jämte språket C)
- Existerande program i Java kan användas i Scala-program
- Twitter, Netflix, Linked-In, Sony, Apple m.fl. använder Scala

Datalogiska grundbegrepp

Några datalogiska grundbegrepp

- **Algoritm**: beskrivning hur man löser ett problem (ett slags recept)
  - **Sekvens**: göra saker steg för steg
  - **Repetition**: upprepa vissa steg många gånger
  - **Alternativ**: välja nästa steg beroende på variablers värde
- **Abstraktion**
  - **Funktion**: en del av ett program som har fått ett namn och som kan återanvändas gång på gång
  - **Objekt**: en del av ett program som sammanför och kapslar in funktioner och variabler
  - **Klass**: kod som beskriver många objekt av samma **typ**
- Uppdragen i Kojo börjar med: sekvens, repetition, funktion, ...

http://community.computingatschool.org.uk/files/4193/original.pdf



http://community.computingatschool.org.uk/files/4193/original.pdf

Computing Progression Pathways

Pupil Progression	Algorithms	Programming & Development	Data & Data Representation	Hardware & Processing	Co
↓	• Understands what an algorithm is and is able to express simple linear (non-branching) algorithms symbolically. (AL) • Understands that computers need precise instructions. (AL) • Demonstrates care and precision to avoid errors. (AL)	• Knows that users can develop their own programs, and can • demonstrate this by creating a simple program in an environment that does not rely on text e.g. programmable robots etc. (AL) • Executes, checks and changes programs. (AL) • Understands that programs execute by following precise instructions. (AL)	• Recognises that digital content can be represented in many forms. (AB) (GE) • Distinguishes between some of these forms and can explain the different ways that they communicate information. (AB)	• Understands that computers have no intelligence and that computers can do nothing unless a program is executed. (AL) • Recognises that all software executed on digital devices is programmed. (AL) (AB) (GE)	• Obt usir • Unc con onli info • Kno con
↓	• Understands that algorithms are implemented on digital devices as programs.(AL) • Designs simple algorithms using loops, and selection i.e. if statements. (AL) • Uses logical reasoning to predict outcomes. (AL) • Detects and corrects errors i.e. debugging, in algorithms. (AL)	• Uses arithmetic operators, if statements, and loops, within programs. (AL) • Uses logical reasoning to predict the behaviour of programs. (AL) • Detects and corrects simple semantic errors i.e. debugging, in programs. (AL)	• Recognises different types of data: text, number. (AB) (GE) • Appreciates that programs can work with different types of data. (GE) • Recognises that data can be structured in tables to make it useful. (AB) (DE)	• Recognises that a range of digital devices can be considered a computer. (AB) (GE) • Recognises and can use a range of input and output devices. • Understands how programs specify the function of a general purpose computer. (AB)	• Nav sim con • Den and way con
↓	• Designs solutions (algorithms) that use repetition and two-way selection i.e. if, then and else. (AL) • Uses diagrams to express solutions. (AB) • Uses logical reasoning to predict outputs, showing an awareness of inputs. (AL)	• Creates programs that implement algorithms to achieve given goals. (AL) • Declares and assigns variables. (AB) • Uses post-tested loop e.g. 'until', and a sequence of selection statements in programs, including an if, then and else statement. (AL)	• Understands the difference between data and information. (AB) • Knows why sorting data in a flat file can improve searching for information. (EV) • Uses filters or can perform single criteria searches for information.(AL)	• Knows that computers collect data from various input devices, including sensors and application software. (AB) • Understands the difference between hardware and application software, and their roles within a computer system. (AB)	• Unc inte wid • Sho ran • Rec una tech
↓	• Shows an awareness of tasks best completed by humans or computers. (EV) • Designs solutions by decomposing a problem and creates a sub-solution for each of these parts. (DE) (AL) (AB) • Recognises that different solutions exist for the same problem. (AL) (AB)	• Understands the difference between, and appropriately uses if and if, then and else statements. (AL) • Uses a variable and relational operators within a loop to govern termination. (AL) (GE) • Designs, writes and debugs modular programs using procedures. (AL) (DE) (AB) (GE) • Knows that a procedure can be used to hide the detail with sub-solution. (AL) (DE) (AB) (GE)	• Performs more complex searches for information e.g. using Boolean and relational operators. (AL) (GE) (EV) • Analyses and evaluates data and information, and recognises that poor quality data leads to unreliable results, and inaccurate conclusions. (AL) (EV)	• Understands why and when computers are used. (EV) • Understands the main functions of the operating system. (DE) (AB) • Knows the difference between physical, wireless and mobile networks. (AB)	• Unc eng are use (EV • Sele serv • Den tech kno
↓	• Understands that iteration is the repetition of a process such as a loop. (AL) • Recognises that different algorithms exist for the same problem. (AL) (GE) • Represents solutions using a structured notation. (AL) (AB) • Can identify similarities and differences in situations and can use these to solve	• Understands that programming bridges the gap between algorithmic solutions and computers. (AB) • Has practical experience of a high-level textual language, including using standard libraries when programming. (AB) (AL) • Uses a range of operators and expressions e.g. Boolean, and applies them in the context of	• Knows that digital computers use binary to represent all data. (AB) • Understands how bit patterns represent numbers and images. (AB) • Knows that computers transfer data in binary. (AB) • Understands the relationship between binary and file size (uncompressed). (AB) • Defines data types: real numbers and Boolean. (AB)	• Recognises and understands the function of the main internal parts of basic computer architecture. (AB) • Understands the concepts behind the fetch-execute cycle. (AB) (AL) • Knows that there is a range of operating systems and application software for the same hardware. (AB)	• Unc sea • Unc pag • Unc digi incl pac

Förslag på lärandeprogression för några datalogiska grundbegrepp

2014-10-24
Björn Regnell, Lunds universitet

Sekvens

Ordningen spelar en helt avgörande roll i vad som händer!

Exempel 1:

```
fram; vänster
fram; höger
fram; vänster
fram; höger
```

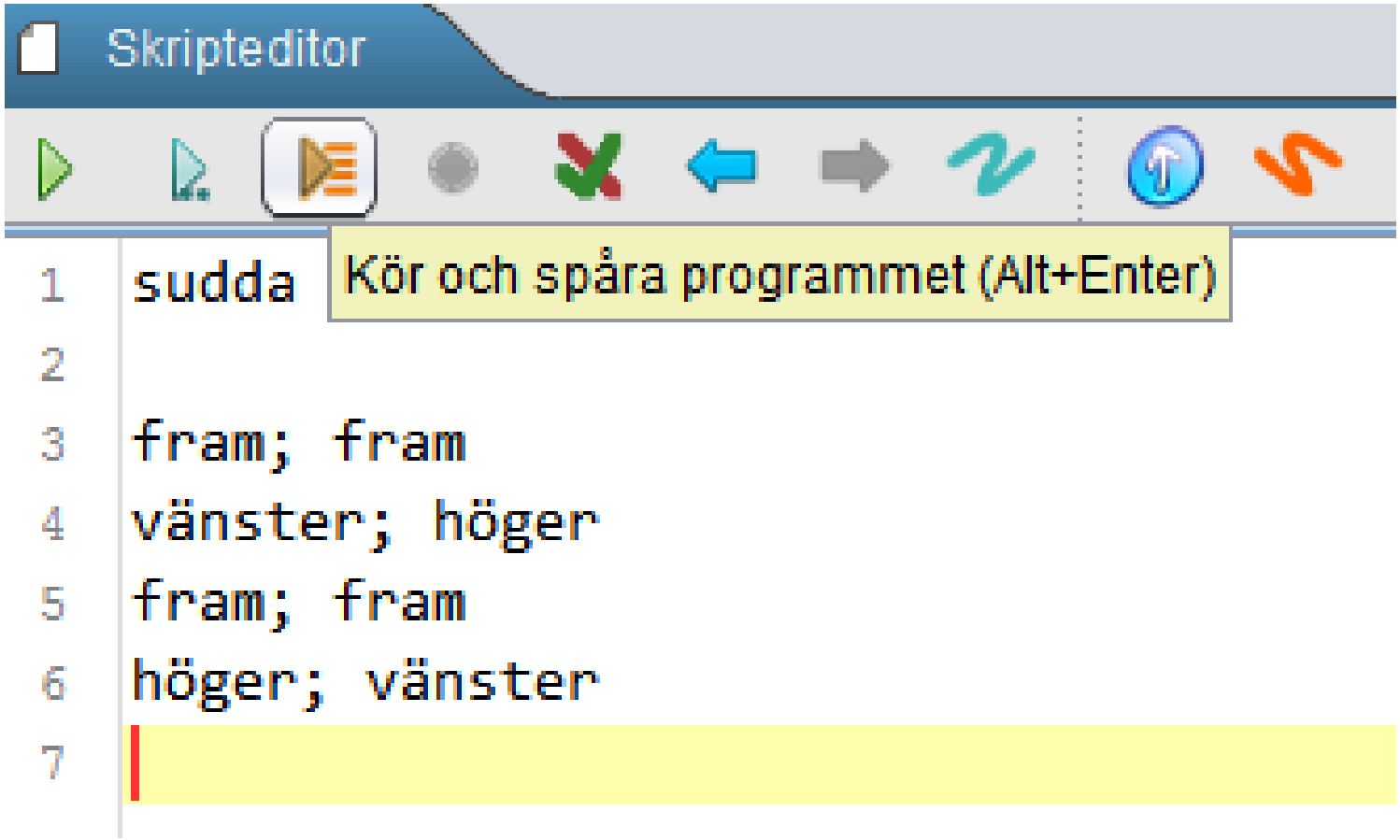
Vad händer om du ändrar ordning?

Exempel 2:

```
fram; fram
vänster; höger
fram; fram
höger; vänster
```

Undersöka sekvens med programspårning

Övning



Repetition (Loop)

– eller ”upprepning” (även kallat ”loop”).

Repetition innebär att datorn upprepar instruktioner om och om igen tills ett villkor (eventuellt) är uppfyllt och repetitionen (eventuellt) avslutas.

Exempel på repetition med förbestämt antal iterationer:

```
upprepa(5000){ utdata("hej") }
utdata("klar!")
```

Det finns också andra sorters repetitioner som vi kommer till senare, där vi inte behöver bestämma på förhand hur många iterationer det blir.

Värde (Value)

– data kan skrivas som olika typer av värden.

Exempel:

```
3
”hej”
3.14
3.0
```

Uttryck (Expression)

– man kan sätta samman värden till sammansatta **uttryck**

Exempel:

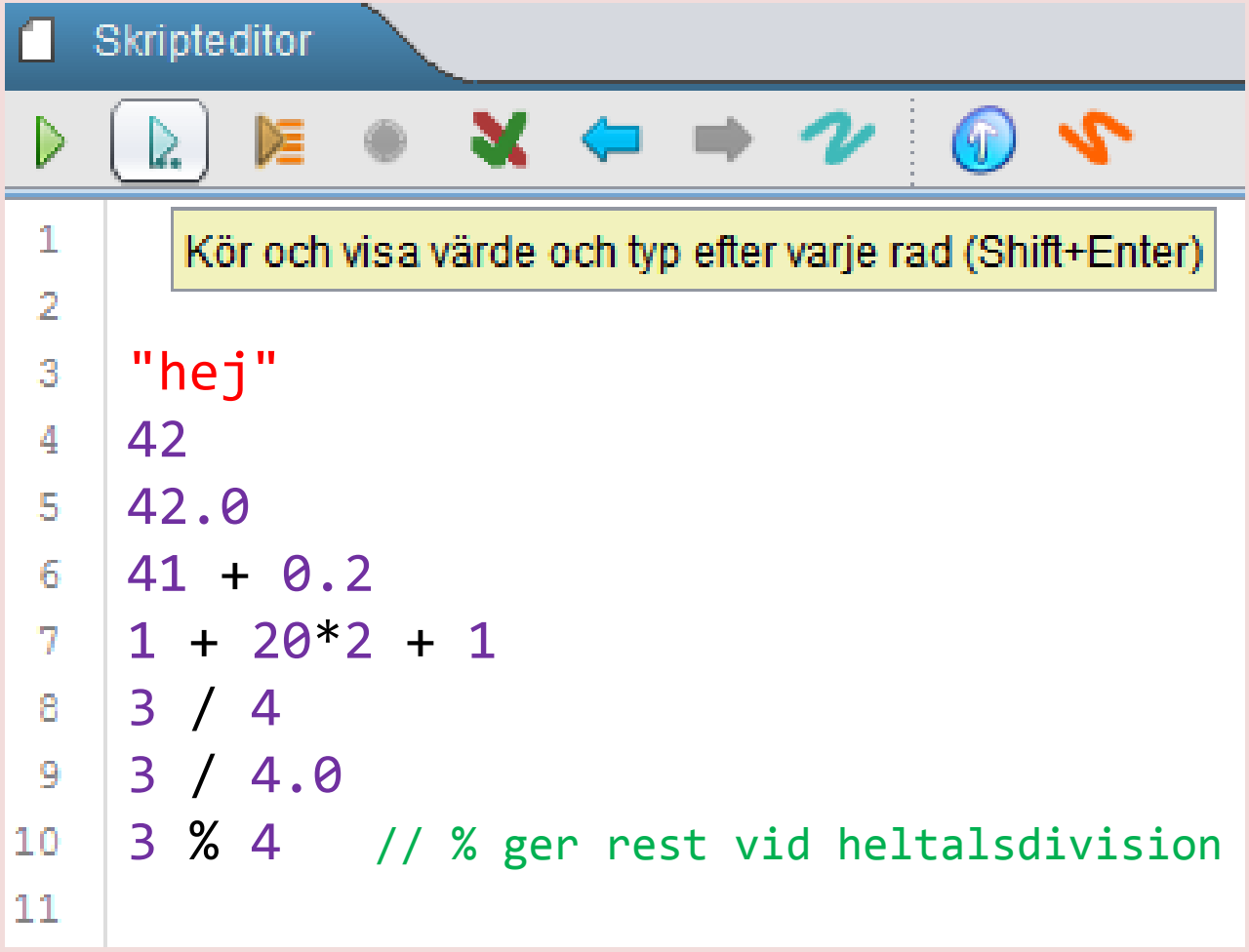
```
10*55+3
"hej"+"san!"
```

Typ (Type)

- Alla värden, variabler, konstanter, parametrar, etc. har en typ.
- Tre vanliga typer:
  - **Int** eller **Heltal**
  - **Double** eller **Decimaltal**
  - **String** eller **Sträng**
- Det finns många fler typer i Scala och Kojo
- Man kan också göra egna nya typer
- Typer anges efter **kolon**
- I Scala slipper man ofta att ange typ eftersom kompilatorn gissar typ om den kan.

Övning

Värde, Uttryck, Typ

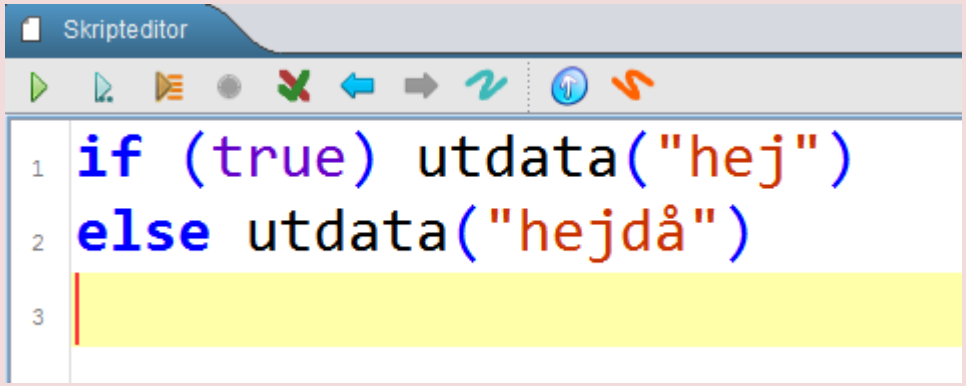


Logiska uttryck

`3 > 1`	har sanningsvärdet **`true`**
`3 < 1`	har sanningsvärdet **`false`**
`&&`	betyder logiskt OCH `(3 > 2) && (3 < 2)` är **`false`**
`||`	betyder logiskt ELLER `(3 > 2) || (3 < 2)` är **`true`**
`==`	kolla om samma värde
`!=`	kolla om olika värde
3 == 3	har sanningsvärdet **`true`**
3 != 3	har sanningsvärdet **`false`**

Övning

Alternativ, sant, falskt



Vad händer om du ändrar **`true`** till **`false`** ?
Vad händer om du skriver `(3 > 1)` efter **`if`** ?
Vad händer om du skriver
`(3 < 1 && 3 > 1)` efter **`if`** ?

Variabel

En variabel är en ”låda” i datorns minne som kan spara data.

En variabel har ett namn och en typ, till exempel heltal.

En variabel deklareras med **var** eller **val**

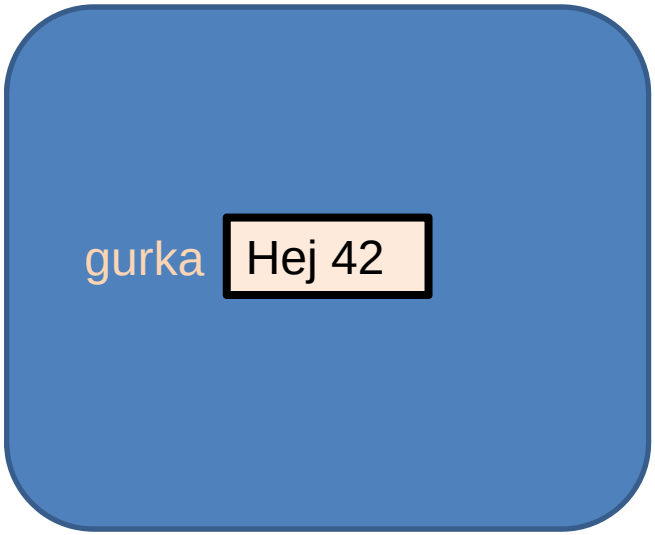
Exempel:

```
var gurka: String = "Hej 42"
```

Namn

Typ

Initialvärde



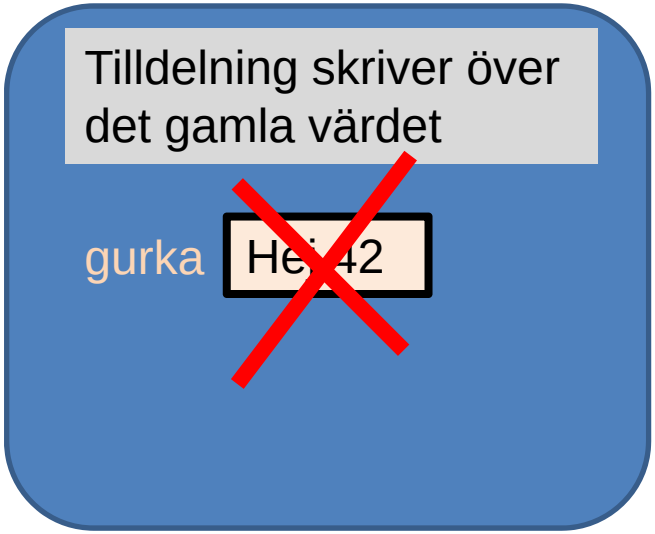
Någonstans i datorns minne

Tilldelningsats

En variabel som är deklarerad med **var** kan ändra värde i en tilldelningssats.

Exempel:

```
gurka = "hejdå"
```



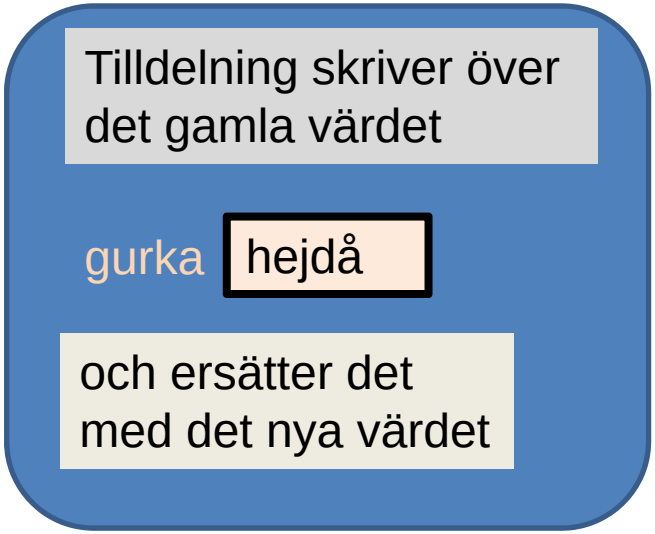
Någonstans i datorns minne

Tilldelningsats

En variabel som är deklarerad med **var** kan ändra värde i en tilldelningssats.

Exempel:

```
gurka = "hejdå"
```



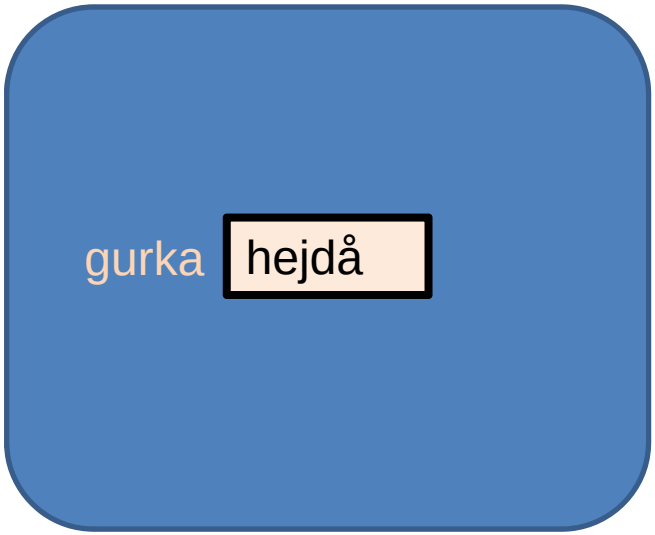
Någonstans i datorns minne

Tilldelningsats

En variabel som är deklarerad med **var** kan ändra värde i en tilldelningssats.

Exempel:

```
gurka = "hejdå"
```



Någonstans i datorns minne

En variabel som är deklarerad med **val** kan inte ändra värde!

Tilldelningstecknet betyder ***inte*** matematisk likhet. För likhet används ==

Indata, utdata, variabel

- Program blir roligare om de kan "prata" med omvärlden.
- Läs in något med `indata` och spara i en variabel och `skriv` ut det igen

```
var s = indata("skriv något: ")
utdata(s)  //skriver i utdatafönstret
skriv(s)   //paddan skriver i ritfönstret
```

Övning

Indata, utdata, variabel

```
var s = indata("skriv något: ")
s = "Du skrev: " + s
utdata(s)
s = s.reverse
skriv(s)
```

Vad händer när du kör koden i Kojo?

Alternativ med if och variabel

Datorn väljer väg i programmet beroende på värdet av ett logiskt uttryck.

```
if (svar == 42) utdata("Grattis!")
else utdata("Försök igen!")
```

Ett logiskt uttryck kan ha värdena **true** eller **false**.

Exempel på logiska uttryck som alla har värdet **true**:

1 == 1

5 >= 4

"hej".size == 3

1 != 0 //ett är inte lika med noll

while-sats

Repetition där antalet iterationer ej är förbestämt

```
// Gissa talet

val s = slumptal(100) + 1
var fortsätt = true
var svar = indata("Gissa ett tal mellan 1 och 100!")
while (fortsätt) {
    if (svar.toInt < s)
      svar = indata("Talet är större, gissa igen!")
    else if (svar.toInt > s)
      svar = indata("Talet är mindre, gissa igen!")
    else if (svar.toInt == s)
      fortsätt = false
}
utdata("Rätt svar!")
```

Gissa talet!

Övning

```
val s = slumptal(100) + 1
var fortsätt = true
var svar = indata("Gissa ett tal mellan 1 och 100! ")
while (fortsätt) {
    if (svar.toInt < s)
        svar = indata("Talet är större, gissa igen!")
    else if (svar.toInt > s)
      svar = indata("Talet är mindre, gissa igen!")
    else if (svar.toInt == s)
      fortsätt = false
}
utdata("Rätt svar!")
```

Uppdrag:

- Inför en variabel **`var antalFörsök = 0`**
  och se till att utskriften på slutet blir
  `Rätt svar! Du klarade det på 5 gissningar!`

Alternativ med match

- Om man har många olika alternativ kan man använda **match**.
- Matchningen sker i tur & ordning uppifrån & ned tills ngt passar.
- Matchning mot understreck passar alla värden - "vad som helst"
- Om inget **case** matchar så avbryts programmet med ett fel.

Exempel:

```
svar match {
       case 41 =>
              utdata("nästan rätt")
       case 42 =>
              utdata("perfekt")
       case 43 =>
              utdata("nästan rätt")
       case _ =>
              utdata("helt fel")
}
```

Man kan kombinera **match** och **if**

```
val s = slumptal(100) + 1
var fortsätt = true
var svar = indata("Gissa ett tal mellan 1 och 100! ")
while (fortsätt) svar match {
    case _ if (svar.toInt < s) =>
      svar = indata("Talet är större, gissa igen!")
    case _ if (svar.toInt > s) =>
      svar = indata("Talet är mindre, gissa igen!")
    case _ => fortsätt = false
}
utdata("Rätt svar!")
```

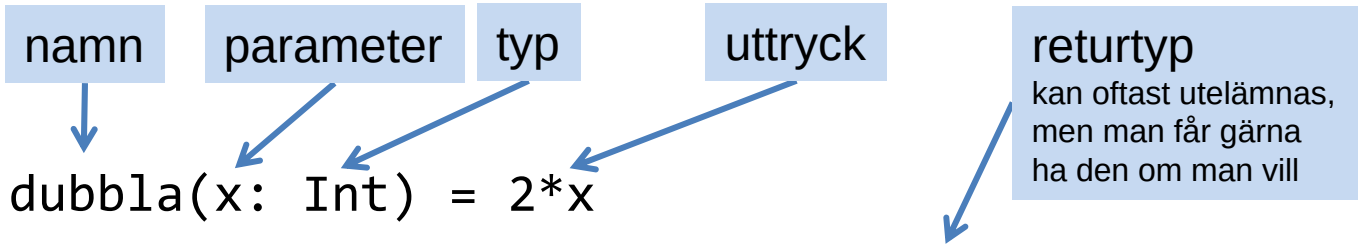
Abstraktion

- En abstraktion ”kapslar in” delprogram som kan återanvändas utan att man behöver veta några detaljer om det som kapslas in.
- En **funktion** med `def` är ett exempel på en abstraktionsmekanism i Scala. Det finns många fler.
- Att kunna använda och skapa abstraktioner är en **central** del av att kunna programmera.
- All programmering innebär att man skapar, kombinerar och återanvänder abstraktioner.
- Utan färdiga abstraktioner i existerande kodbibliotek hade vi behövt uppfinna hjulet varje gång...

Funktion och Parameter

– en funktion är ett ”byggblock” av instruktioner som exekveras när funktionen anropas.

En funktion kan ha ett namn, många parametrar och ett uttryck som resulterar i ett resultatvärde.




```
def dubbla(x: Int) = 2*x
def plus(a: Heltal, b: Heltal): Heltal = a + b

def multipliceraOchSkriv(x: Int, y: Int) = {
      val resultat = x * y
      skriv(resultat)
      resultat
}
```

Sista uttrycket i blocket blir funktionens **returvärde**

Argument

– när man anropar en funktion behövs ett argument till varje parameter. Parameterns namn byts ut mot argumentet. Parameternamnet är lokalt för funktionen.

```
def öka(x: Int) = x+1
```

Exempel på ett funktionsanrop med argumentet 41:

`öka(41)` byts ut mot `41+1`

räknas ut till `42`

`42` blir funktionens returvärde

Övning

Funktion, parameter, argument

```
def öka(x: Int) = x+1

var x = 41
utdata(öka(x))
utdata(x)
x = öka(öka(x))
utdata(x)
```

Vad kommer att skrivas ut?
Försök förklara vad som händer.

Skillnaden mellan def, val och var

- `def minFunktion = 41 + 1`
  - En **funktion** som sätter namn på en eller flera kodrader
  - Varje gång minFunktion **anropas** så körs koden som funktionen definierar
  - En funktion som inte lämnar något värde kallas för **procedur** eller kommando, t.ex.:
    ```
    def minProcedur { skriv(41+1) }
    ```
- `val minKonstant = 41 + 1`
  - Efter att en konstant fått sitt initialvärde så ändras den aldrig igen (val efter engelskans value)
- `var minVariabel = 41 + 1`
  - En variabel som deklareras med var kan ändra värde genom en tilldelningssats, som räknar ut ett nytt värde och skriver över det gamla värdet med det nya värdet:

```
minVariabel = minVariabel + 1
```

Objekt, medlem och metod

Ett **objekt** för samman variabler och funktioner som hör ihop.

```
object mittKonto {
  var saldo = 0
  def in(belopp: Int) {saldo = saldo + belopp}
  def ut(belopp: Int) {saldo = saldo - belopp}
}
```

Man kan komma åt medlemmar med *punktnotation*:

```
utdata(mittKonto.saldo)
mittKonto.in(100)
mittKonto.ut(58)
utdata(mittKonto.saldo)
```

Funktionerna och variablerna som finns i objektet kallas *medlemmar*.

Funktioner som är medlemmar kallas *metoder*.

Övning

Använd `object timer`

```
object timer {
  def nu = System.currentTimeMillis  //ger nutid i millisekunder
  var tid = nu
  def nollställ { tid = nu }
  def mät = nu - tid
  def slumpvänta(min: Int, max: Int) =
    Thread.sleep((max-min)*slumptal(1000)+min*1000)
}
```

Använd timer för att göra detta program:

- Skriv ut texten "Vänta..."
- Vänta en slumpmässig tid om minst 2 och max 6 sekunder
- Nollställ tidtagaren
- Läs indata med ledtexten "Tryck Enter så snabbt du kan."
- Skriv ut reaktionstiden i sekunder

- Extrauppgift: Googla på "java currentTimeMillis". Från när räknas "nutiden"?

Lösning

Använd `object timer`

```
object timer {
  def nu = System.currentTimeMillis  //ger nutid i millisekunder
  var tid = nu
  def nollställ { tid = nu }
  def mät = nu - tid
  def slumpvänta(min: Int, max: Int) =
    Thread.sleep((max-min)*slumptal(1000)+min*1000)
}

utdata("Vänta...")
timer.slumpvänta(2,6)
timer.nollställ
indata("Tryck Enter så snabbt du kan.")
utdata("Reaktionstid: " + timer.mät/1000.0 + " sekunder")
```

Hur gör vi om vi vill ha flera objekt av samma typ?

Det finns än så länge bara ett enda konto i vår bank:

```
object mittKonto {
  var saldo = 0
  def in(belopp: Int) {saldo = saldo + belopp}
  def ut(belopp: Int) {saldo = saldo - belopp}
}
```

Singelobjektet `mittKonto` skapas med ovan kod **i ett enda exemplar**.

Men en riktig bank har ju mer än en kund...

Klass

En **klass** används för att skapa många olika objekt av samma typ.

```
class Konto {
  var saldo = 0
  def in(belopp: Int) {saldo = saldo + belopp}
  def ut(belopp: Int) {saldo = saldo - belopp}
}
```

Man skapar nya objekt med **new**:

```
var konto1 = new Konto
var konto2 = new Konto

konto1.in(100)
konto2.in(200)
```

Klassparameter

Klasser kan ha parametrar.

Konton i en bank borde ju kunna ha olika kontonummer:

```
class Konto(nummer: Int) {
  var saldo = 0
  def in(belopp: Int) {saldo = saldo + belopp}
  def ut(belopp: Int) {saldo = saldo - belopp}
  def visaSaldo {utdata("Konto: " + nummer + " Saldo: " + saldo)}
}
```

Klassparametrarna tilldelas värden när objekten skapas:

```
var konto1 = new Konto(1234)
var konto2 = new Konto(4567)

konto1.in(100)
konto2.in(200)
konto1.visaSaldo
konto2.visaSaldo
```

Datastruktur, samling (collection)

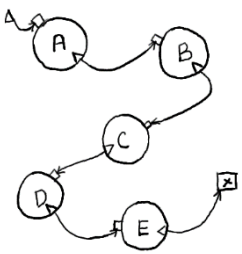
- När datorn bearbetar stora datamängder lagras data i olika typer av strukturer i minnet som gör att det går snabbt att hitta och bearbeta specifika dataelement.
- En samling är en sorts datastruktur som kan lagra **variabelt antal** värden (noll eller flera) av **samma typ**.
- Exempel: Vektor med heltal

plats	0	1	2	3	4	5	6
innehåll	42	21	13	12	31	12	42

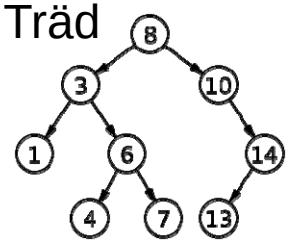
Matris (Vektor med vektorer)

	X/Y	1	2	3	4	5	6	7	8	9	10	total
1	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	3	0.00	10.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.61
4	4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	5	0.00	0.00	79.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	79.43
6	6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	8	0.00	0.00	0.00	0.00	0.00	0.00	73.41	0.00	0.00	0.00	73.41
9	9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	Sum	0.00	10.61	79.43	0.00	0.00	0.00	73.41	0.00	0.00	0.00	163.45

Länkad lista



Träd



Övning

Vector

```
val ord = Vector("hej", "på", "dej")
utdata(ord)

val första = ord(0)
utdata(första)

val stora = ord.map(x => x.toUpperCase)
utdata(stora)
```

Prova programmet ovan. Vad händer?

Vektorer: Programmeringens "arbetshästar"

```
val xs = Vector(1, 2, 3, 4)     // xs: Vector[Int] = Vector(1, 2, 3, 4)

val huvud = xs.head             // huvud: Int = 1

val svans = xs.tail             // svans: Vector[Int] = Vector(2, 3, 4)

val omvänt = xs.reverse         // omvänt: Vector[Int] = Vector(4, 3, 2, 1)

val svansOmv = xs.tail.reverse // svansOmv: Vector[Int] = Vector(4, 3, 2)

val dubblad = xs.map(_ * 2)     // dubblad: Vector[Int] = Vector(2, 4, 6, 8)
```

- Strängar med text är en slags vektor av bokstäver och många operationer som fungerar på Vector fungerar också på String:

```
"svensk musikgrupp från 80-talet".reverse
```

- Övning: Prova denna funktion som kollar om en sträng är en palindrom:

```
def ärPalindrom(s: String) = { s.reverse == s }
utdata(ärPalindrom("ABBA"))
```

En oerhört användbar datastruktur: Uppslagstabell med samlingen `Map`

- Skapa en `Map` som gör att du snabbt hittar rätt värde givet en nyckel.
- Liknar vektor, men "platsindex" kan vara annat än heltal.
- Används för databaser, uppslagstabeller.

```
val huvudstad = Map(
  "Danmark" -> "Köpenhamn",
  "Norge" -> "Oslo",
  "Finland" -> "Helsingfors",
  "Skåne" -> "Malmö"
)

println( huvudstad("Skåne") )
```

- Extrauppgift: läs mer här http://sv.wikipedia.org/wiki/Hashtabell

for-sats i Scala

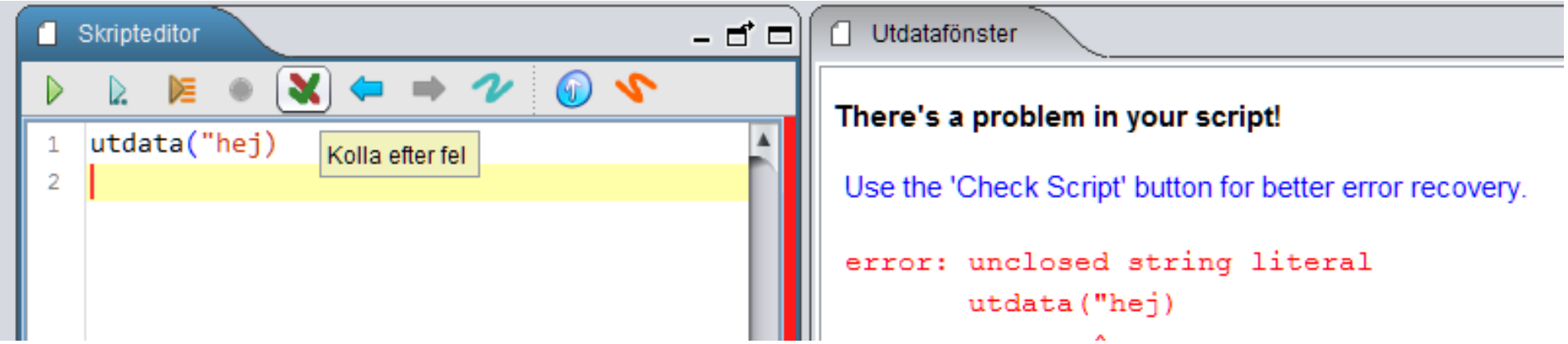
```
for (i <- 1 to 10) { print(i + " ") }

val xs = Vector("a", "b", "c")

val ys = for (x <- xs) yield x+x
```

Programmeringstips

Att tolka felmeddelande

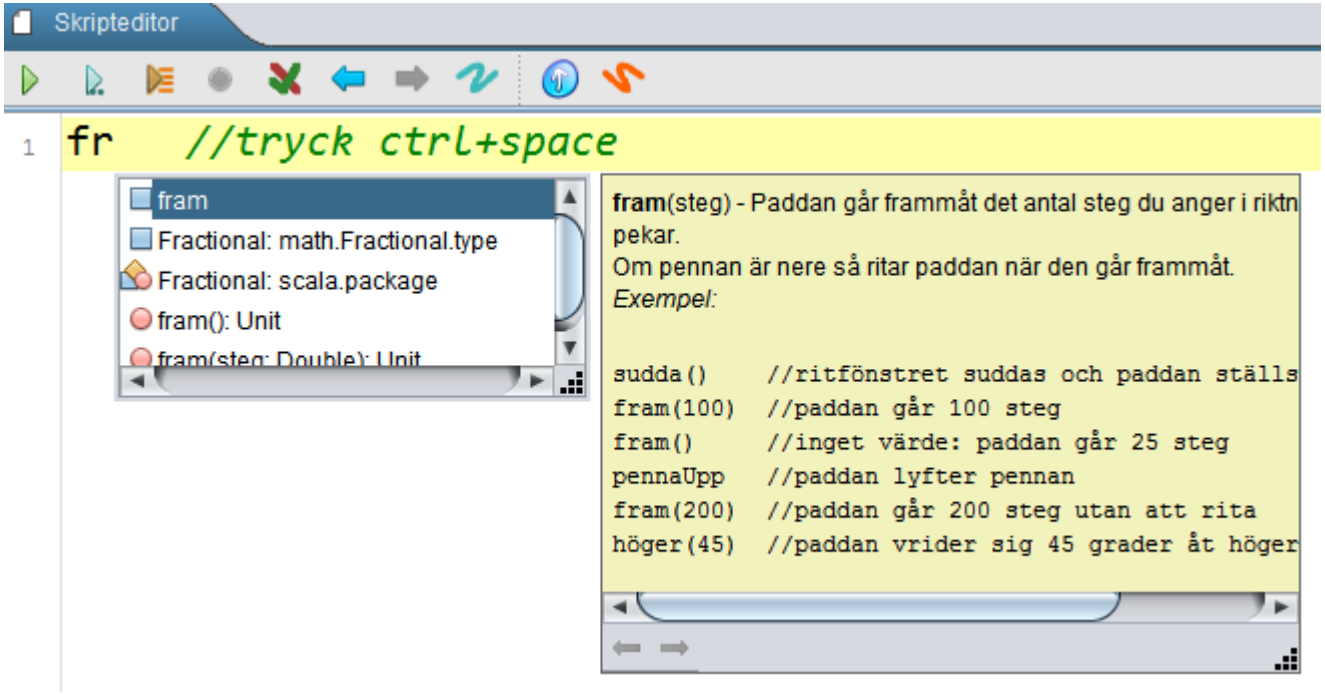


- Ofta är felmeddelande kryptiska men försök
  lära dig vad olika typer av fel **brukar** ge för felmeddelande
- Klicka på den röda understrykningen
  för att hoppa dit felet (kanske) är
- Tryck på "**kolla-efter-fel**"-knappen
  så kanske du får reda på mer om felet och var det är
- Prova dig fram genom att **ta bort kod tills det fungerar**
- Med bakåtpilen får du tillbaka dina gamla program (om har tryckt play)
- Om det **saknas parenteser och krullparenteser** är det svårt för
  kompilatorn att peka ut var felet är
  och felmeddelandet kan bli helknasigt

Upptäck funktioner som finns i Kojo med *kodkomplettering*

- för att se allt som börjar på ...

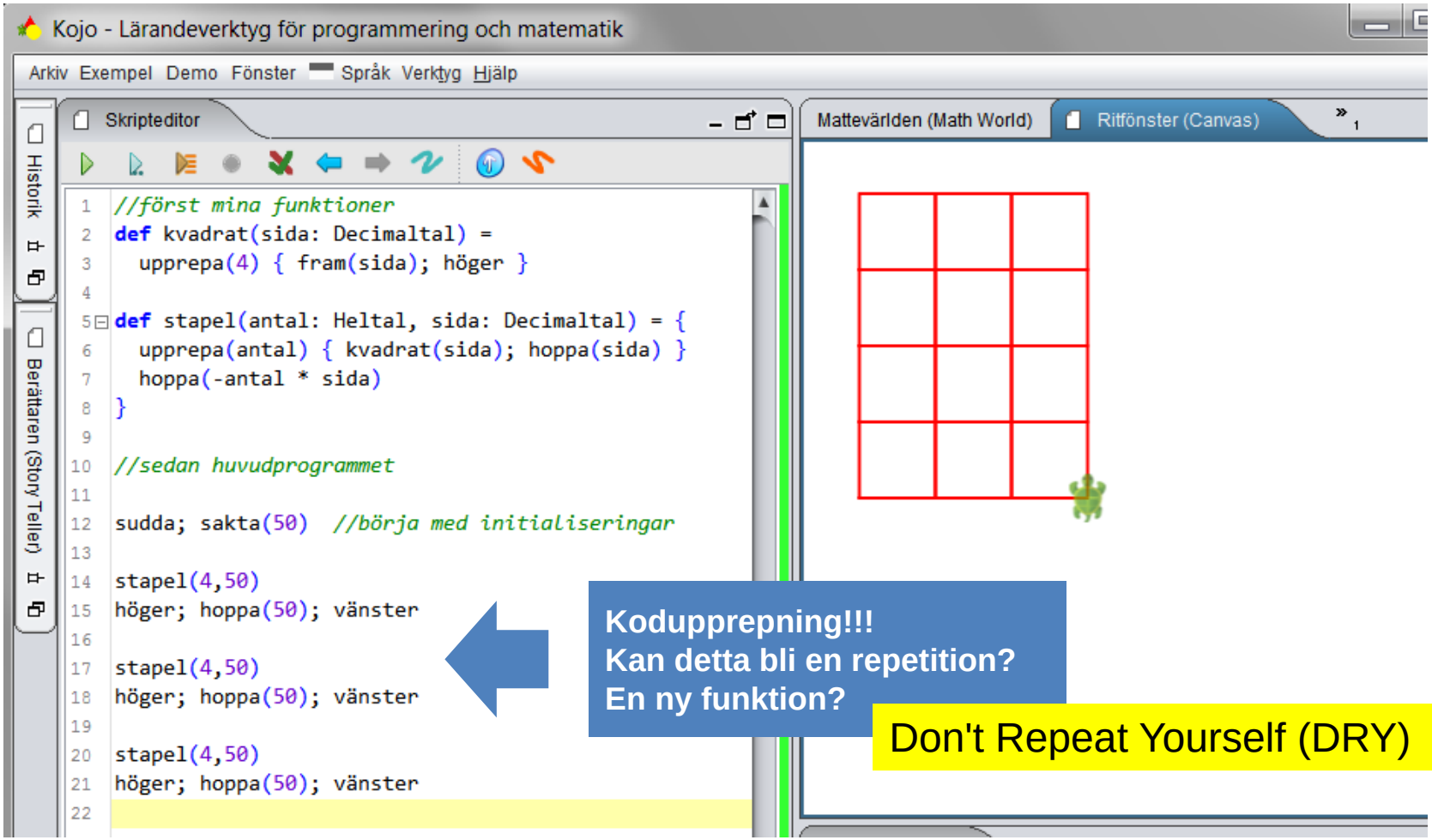
  fr***<Ctrl+Space>*** *eller <Ctrl+Alt+Space>*



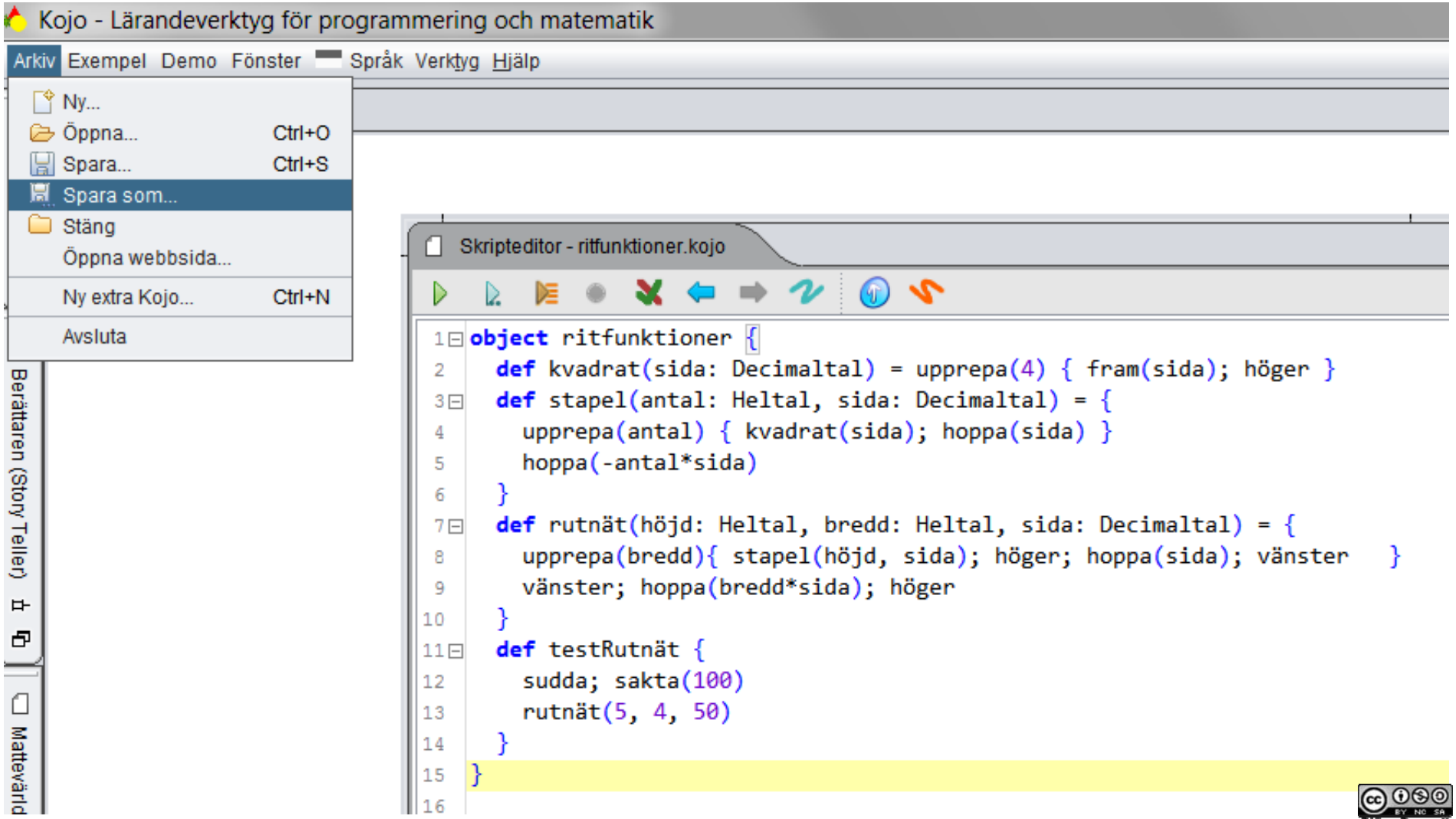
Tryck Esc för att bli av med listan

För att ctrl+space ska fungera på mac:
Ändra Systeminställningar -> Spotlight -> kortkommando för Spotlight-menyn -> ⌘ Mellanslag

Skapa en bra kodstruktur genom att leta abstraktionsmöjligheter



Lägg dina funktioner i ett objekt och spara objektet i en egen fil

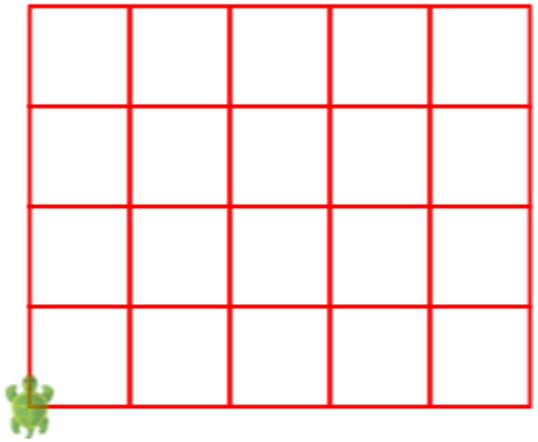


Använd dina funktionsbibliotek lättare med hjälp av **import**

Skripteditor - ritfunktioner.kojo*

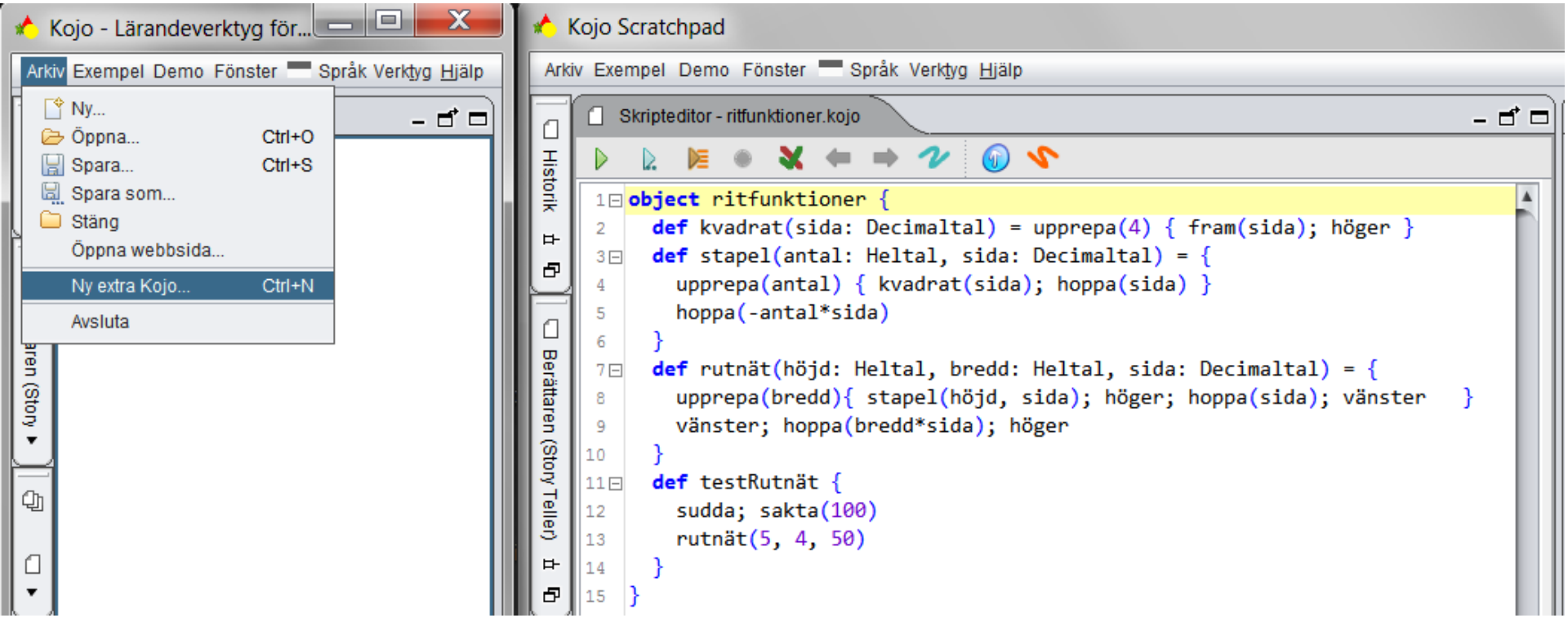
```
1  object ritfunktioner {
2    def kvadrat(sida: Decimaltal) =
3      upprepa(4) { fram(sida); höger }
4
5    def stapel(antal: Heltal, sida: Decimaltal) = {
6      upprepa(antal) { kvadrat(sida); hoppa(sida) }
7      hoppa(-antal * sida)
8    }
9
10   def rutnät(höjd: Heltal, bredd: Heltal, sida: Decimaltal) = {
11     upprepa(bredd) { stapel(höjd, sida); höger; hoppa(sida); vänster }
12     vänster; hoppa(bredd * sida); höger
13   }
14   def testRutnät {
15     sudda; sakta(50)
16     rutnät(4,5,50)
17   }
18 }
19
20 import ritfunktioner._
21 testRutnät
```

Ritfönster (Canvas)



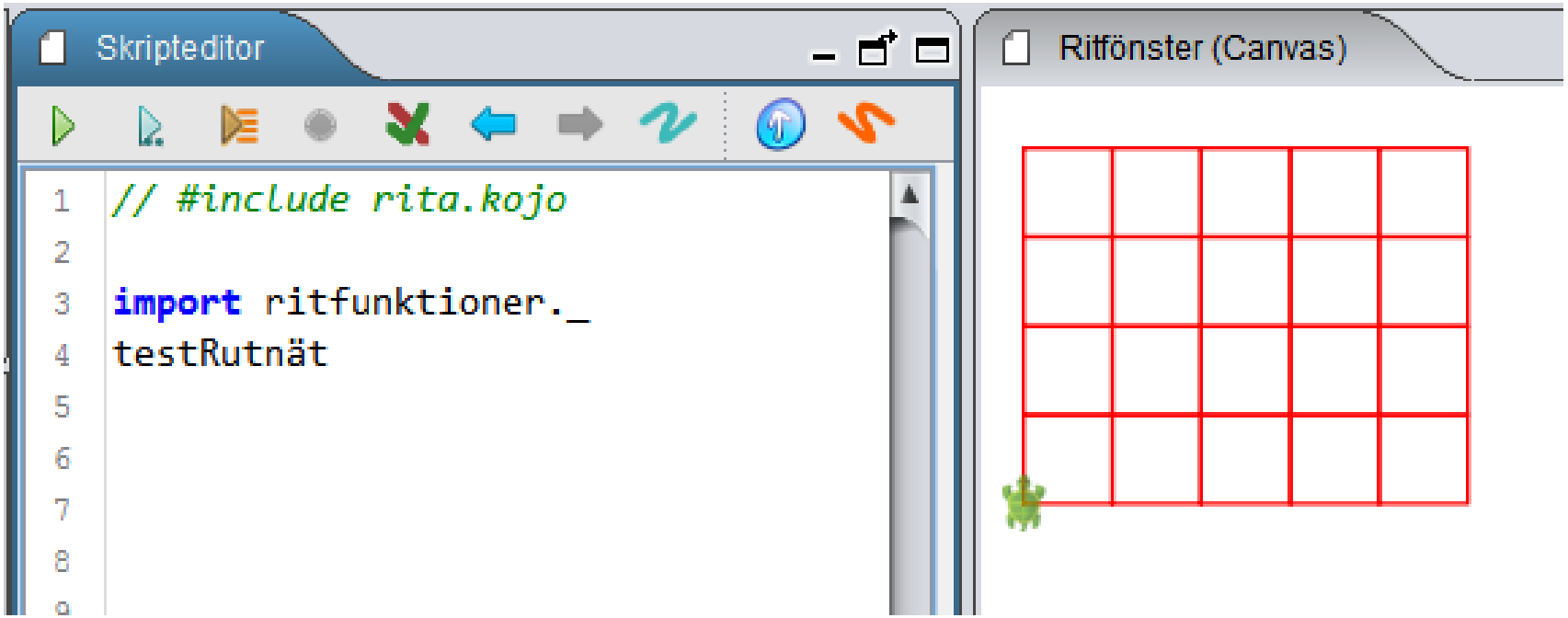
Import gör så att du kommer åt objektets medlemmar utan punktnotation

Ha flera Kojo igång samtidigt



Jobba parallellt i flera Kojo-fönster
med att utveckla hjälpfunktioner och att testa dem.

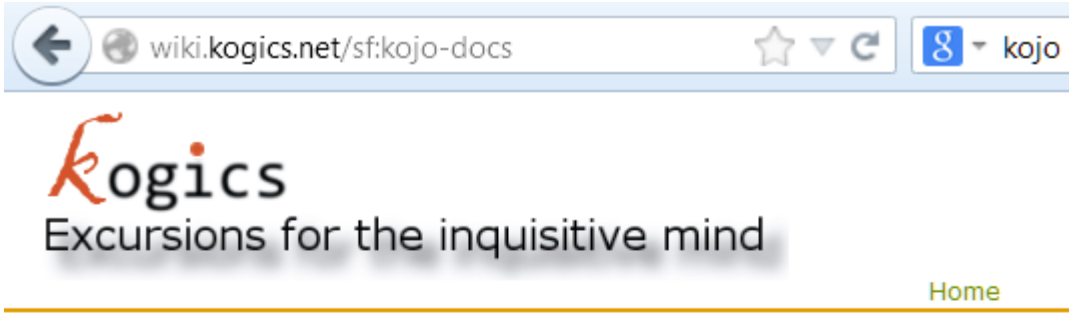
Kodstruktur med `//#include`



Spara ditt funktionsbibliotek i en egen fil som heter exempelvis **rita.kojo**
Gör så att innehållet i filen laddas genom att först i programmet skriva:

```
// #include rita.kojo
```

Mer studiematerial (på engelska)



The Kojo Wiki

Core Learning material

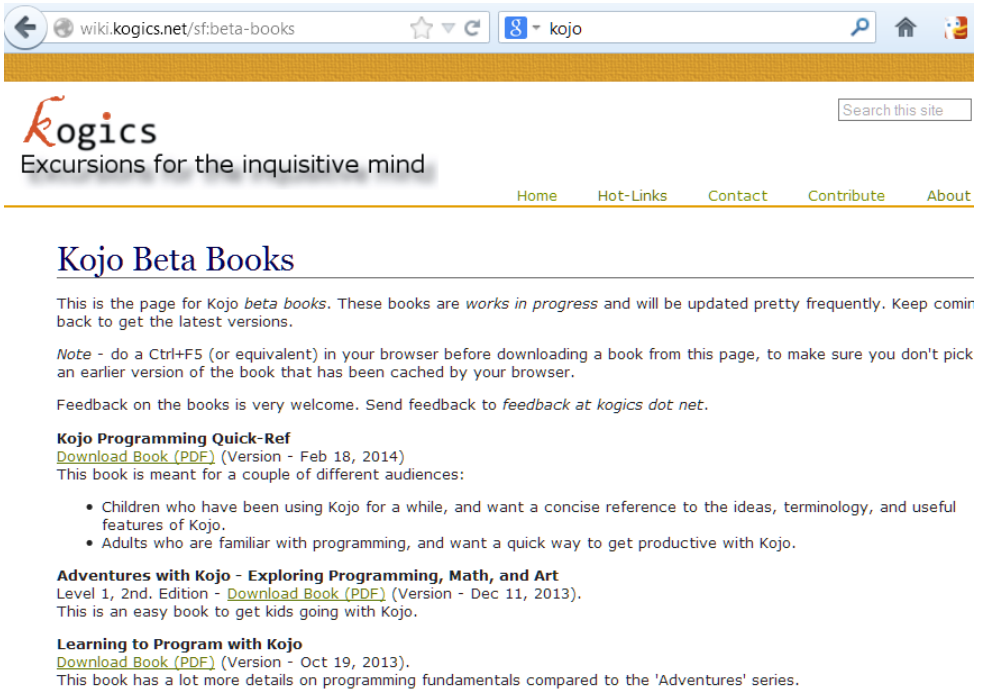
- Getting started with Kojo
- Kojo Ebooks
- Kojo Beta Ebooks
- Pictures in Kojo (*New*)
- Troubleshooting

Topics of Interest

- Kojo FAQ
- Stories for Learning
- Contributing to Kojo
- Kojo in Education

Reference material

- Turtle, Canvas, and Utility Commands Reference
- Staging Reference, and Additional Staging Material



Kojo Beta Books

This is the page for Kojo *beta books*. These books are *works in progress* and will be updated pretty frequently. Keep comin back to get the latest versions.

Note - do a Ctrl+F5 (or equivalent) in your browser before downloading a book from this page, to make sure you don't pick an earlier version of the book that has been cached by your browser.

Feedback on the books is very welcome. Send feedback to *feedback at kogics dot net*.

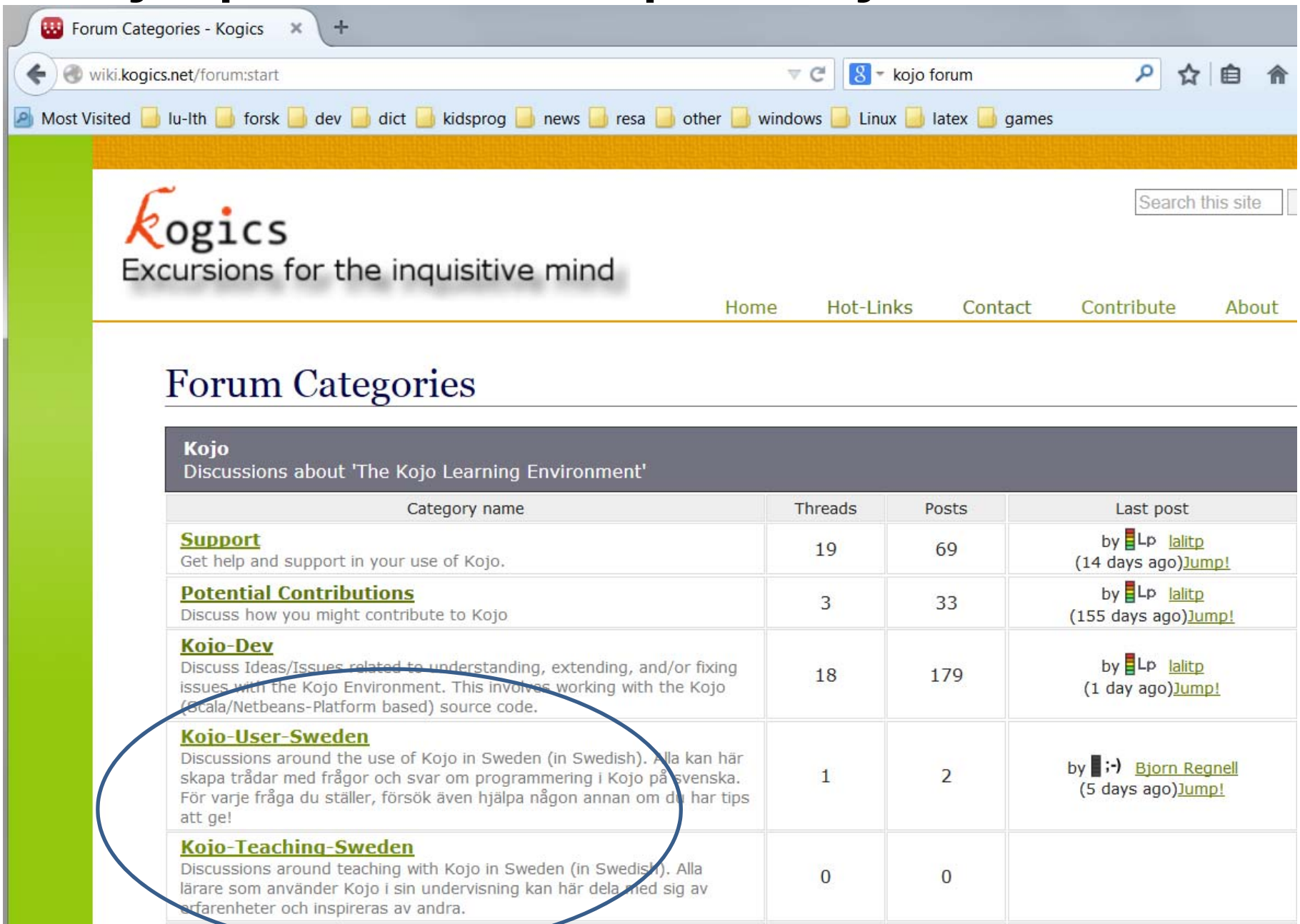
Kojo Programming Quick-Ref
Download Book (PDF) (Version - Feb 18, 2014)
This book is meant for a couple of different audiences:

- Children who have been using Kojo for a while, and want a concise reference to the ideas, terminology, and useful features of Kojo.
- Adults who are familiar with programming, and want a quick way to get productive with Kojo.

Adventures with Kojo - Exploring Programming, Math, and Art
Level 1, 2nd. Edition - Download Book (PDF) (Version - Dec 11, 2013).
This is an easy book to get kids going with Kojo.

Learning to Program with Kojo
Download Book (PDF) (Version - Oct 19, 2013).
This book has a lot more details on programming fundamentals compared to the 'Adventures' series.

Hjälp varandra på Kojos forum!



www.geogebra.org/cms/sv/

geogebra

Swedish

GeoGebra ned Hjälp Material Forum

Home Blog Info Roadmap Team ToS

GeoGebra

Dynamic mathematics & science for learning and t

Software

Materials

- From elementary school to university level
- Interactive geometry, algebra, statistics, and ca
- Tens of thousands of free materials

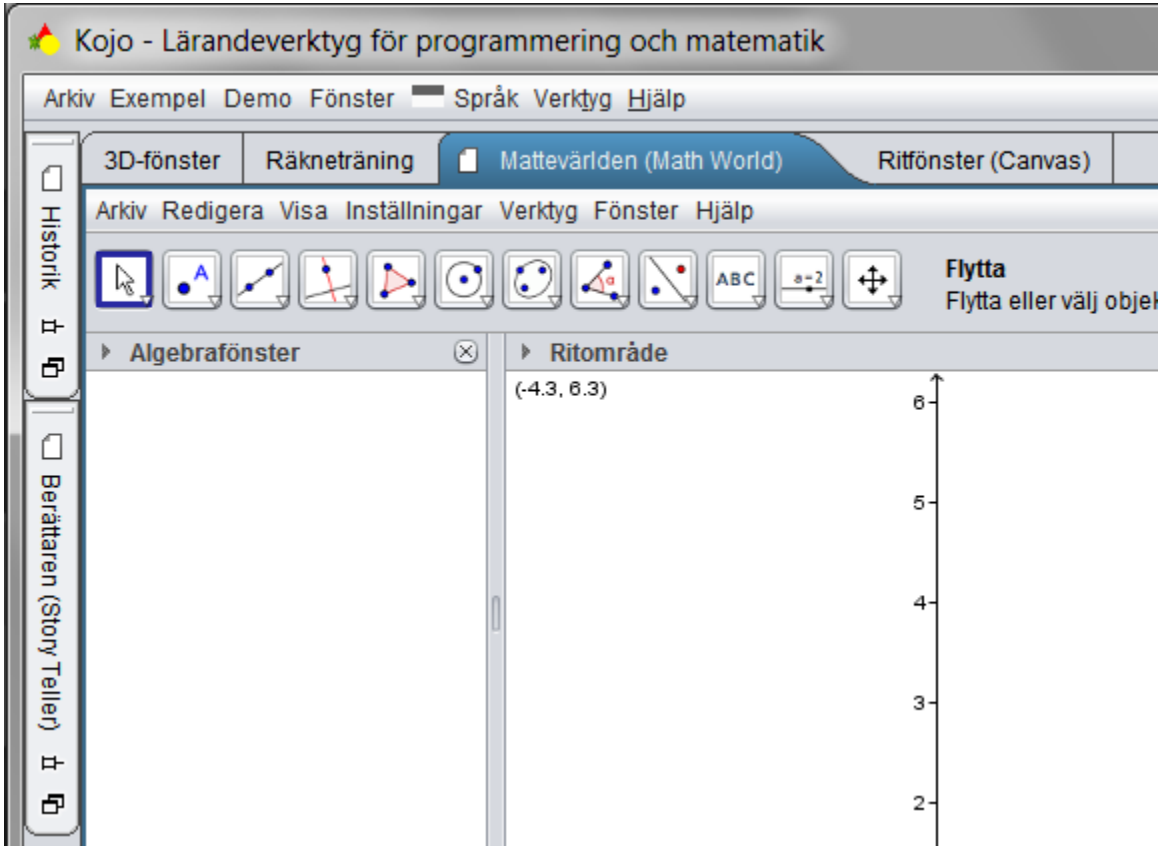
News

GeoGebra 4.4

Try GeoGebra 4.4 with faster and better CAS view!

www.geogebra.org

Pedagogiskt verktyg för geometri och algebra finns i Kojo i "Mattevärlden"-fönstret



Avslutning

Förslag på lärandeprogression för några datalogiska grundbegrepp

Avancera djupare

Avancera bredare

Sekvens

Repetition

Abstraktion

Data

Alternativ

Satser i sekvens, ny rad, semikolon ; ordningen spelar roll

Block { }

Block med lokala namn

Nästlade block

Överskuggning av namn

Repetition som tar medlemmar ur en samling
for (x <- xs) {...}

Repetition med loopvariabel utan förbestämt antal iterationer, while

Repetitionsuttryck
for(...) yield {...}

Repetition utan loopvariabel med bestämt antal iterationer "upprepa"

Definiera egna funktioner utan parameter, def

Nästlad repetition, förstå att antalet iterationer multipliceras

Använda funktioner utan parameter

Använda funktioner med parameter ( )

Definiera egna funktioner med parameter

Repetition med loopvariabel & bestämt antal iterationer,
for (...) { }

Använda högre ordningens funktioner

Skillnaden mellan val, lazy val och def

Aktiveringspost, anropsstack

Värde

Uttryck

Standardtyper, kolon :
Int, Double, String
Heltal, Decimaltal, Sträng

Definiera funktioner som returnerar värde

Mönster-matchning

Rekursiva funktioner

Alternativ om sant
if (true) ...
else ...

Oföränderliga samlingar
Vector, Set, Map

Klasser class

Skillnad m. värdeanrop & namnanrop

Alternativ om falskt
if (false) ...
else ...

Använda sanningsvärden och logiska uttryck

Singelobjekt som samlar funktioner
object

Ärvning, extends

Referensvariabler

Enskaper & inmixning, trait

Konstant, val

Alternativ med variabel

Medlemsåtkomst, punktnotation

case-klasser

Definiera högre ordningens funktioner

Variabel, var

Tupler (1,"hej")

Polymorfism & dynamisk bindning

Alternativ med
match ...
case =>

Skillnaden mellan föränderliga och oföränderliga värden

Föränderliga samlingar, Array

Referenslikhet
Strukturlikhet

Deklaration, initiering, tilldelning av variabel =

Singelobjekt som samlar både variabler och funktioner,
private

App, main

Samtidighet, trådar, händelser

Skillnad m. referenstyper och värdetyper

Kodfiler, package

2014-10-24
Björn Regnell, Lunds universitet

Några vanliga trösklar i datalogiskt tänkande

- Exekveringsmodell:
  sekvens, repetition, alternativ, abstraktion
- Variabler och tilldelning
- Parametrar som byts ut mot argument
- Sanningsvärden och logiska uttryck
- Kombinera flera abstraktioner
- Lokala namn i funktioner och block
- Felsökningsmetod, prova sig fram:
  ta bort, ändra, skriv ut delresultat, fråga

Sammanfattning

1. Kännedom om några grundläggande principer i datalogiskt tänkande
   - sekvens, repetition, abstraktion, alternativ, variabel, objekt, klass, ...
2. Exempel på progression i datalogiskt tänkande
3. Kännedom om några lärandetrösklar
4. Praktiska programmeringstips