EDAA45 Programmering, grundkurs

Läsvecka 1: Introduktion

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

1 Introduktion

- Om kursen
- Att lära denna läsvecka `w01`
- Om programmering
- De enklaste beståndsdelarna: litteraler, uttryck, variabler
- Funktioner
- Logik
- Satser

Om kursen

Nytt för i år 2016

- **Scala** införs som förstaspråk på Datateknikprogrammet.
- Den **största förnyelsen** av den inledande programmeringskursen sedan vi införde **Java 1997**.
  - Nya föreläsningar
  - Nya övningar
  - Nya laborationer
  - Nya skrivningar
- Allt kursmaterial är **öppen källkod**.
- **Studentermedverkan** i kursutvecklingen.

www.lth.se/nyheter-och-press/nyheter/visa-nyhet/article/scala-blir-foerstaspraak-paa-datateknikprogrammet/

Veckoöversikt

W	*Modul*	*Övn*	*Lab*
W01	Introduktion	expressions	kojo
W02	Kodstrukturer	programs	–
W03	Funktioner, objekt	functions	blockmole
W04	Datastrukturer	data	pirates
W05	Sekvensalgoritmer	sequences	shuffle
W06	Klasser	classes	turtlegraphics
W07	Arv	traits	turtlerace-team
KS	KONTROLLSKRIVN.	–	–
W08	Mönster, undantag	matching	chords-team
W09	Matriser, typparametrar	matrices	maze
W10	Sökning, sortering	sorting	survey
W11	Scala och Java	scalajava	Ithopoly-team
W12	Trådar, webb	threads	life
W13	Design, api	Uppsamling	Projekt
W14	Tentaträning	Extenta	–
T	TENTAMEN	–	–

Vad lär du dig?

- Grundläggande principer för programmering:
  Sekvens, Alternativ, Repetition, Abstraktion (SARA)
  $\Longrightarrow$ Inga förkunskaper i programmering krävs!
- Implementation av algoritmer
- Tänka i abstraktioner, dela upp problem i delproblem
- Förståelse för flera olika angreppssätt:
  - **imperativ programmering**
  - **objektorientering**
  - **funktionsprogrammering**
- Programspråken **Scala** och **Java**
- Utvecklingsverktyg (editor, kompilator, utvecklingsmiljö)
- Implementera, testa, felsöka

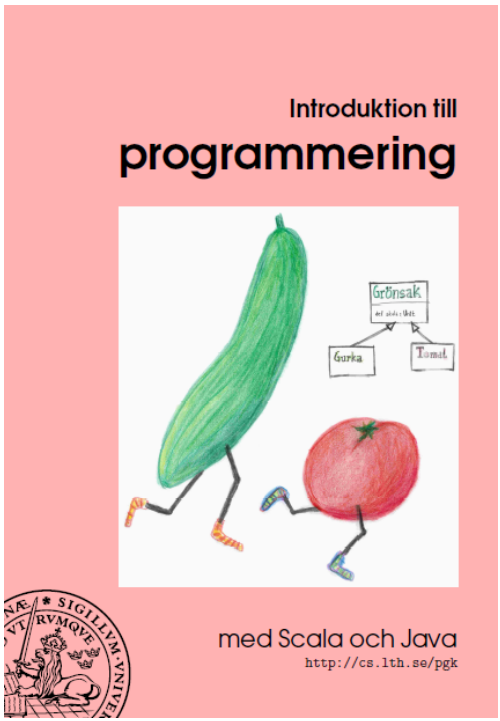
Varför Scala + Java som förstaspråk?

- Varför Scala?
  - Enkel och enhetlig syntax => lätt att skriva
  - Enkel och enhetlig semantik => lätt att fatta
  - Kombinerar flera angreppsätt => lätt att visa olika lösningar
  - Statisk typning + typhärledning => färre buggar + koncis kod
  - Scala Read-Evaluate-Print-Loop => lätt att experimentera
- Varför Java?
  - Det mest spridda språket
  - Massor av fritt tillgängliga kodbibliotek
  - Kompabilitet: fungerar på många platformar
  - Effektivitet: avancerad & mogen teknik ger snabba program
- Java och Scala fungerar utmärkt tillsammans
- Illustrera likheter och skillnader mellan olika språk
  => Djupare lärande

Hur lär du dig?

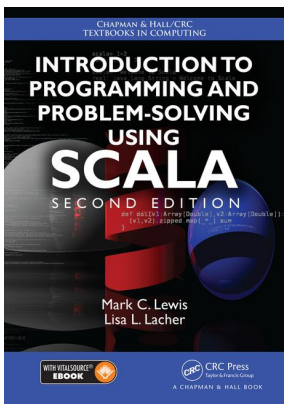
- Genom praktiskt **eget arbete**: **Lära genom att göra!**
  - Övningar: applicera koncept på olika sätt
  - Laborationer: kombinera flera koncept till en helhet
- Genom studier av kursens teori: **Skapa förståelse!**
- Genom samarbete med dina kurskamrater: **Gå djupare!**

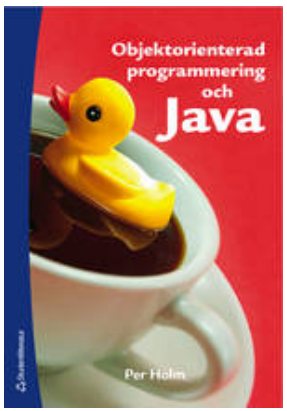
Kurslitteratur



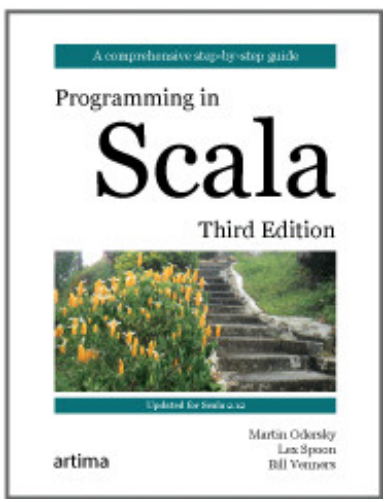
- **Kompendium** med övningar & laborationer, trycks & säljs av inst. på beställning
- Föreläsningsbilder
- Nätresurser enl. länkar

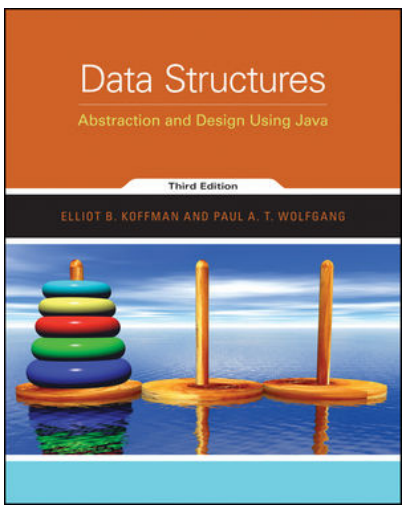
Bra, men ej nödvändig, **bredvidläsning**:
– för **nybörjare**:





– för de som **redan kodat** en del:





Beställning av kompendium och snabbreferens

- **Kompendiet** finns i pdf för fri nedladdning enl. CC-BY-SA, men det **rekommenderas starkt** att du köper den tryckta bokversionen.
- Det är mycket lättare att ha övningar och labbar **på papper bredvid skärmen**, när du ska tänka, koda och plugga!
- **Snabbreferensen** finns också i pdf men du behöver ha en tryckt version eftersom det är **enda tillåtna hjälpmedlet** på skriftliga kontrollskrivningen och tentamen.
- Kompendiet och snabbreferens trycks här i E-huset och säljs av institutionen till **självkostnadspris**.
- Pris för kompendium **beror på hur många som beställer**.
- Snabbreferens kostar 10 kr.
- Skriv upp dig på listan som skickas runt – tryckning enligt beställning.
- Du betalar **kontant** med **jämna pengar** på cs expedition, våning 2.

Föreläsningsanteckningar

- Föreläsningbilder utvecklas under kursens gång.
- Alla bilder läggs ut här: github.com/lunduniversity/introprog/tree/master/slides och uppdateras kontinuerligt allt eftersom de utvecklas.
- Förslag på innehåll välkomna!

Personal

Kursansvarig:

Björn Regnell, bjorn.regnell@cs.lth.se

Kurssekreterare:

Lena Ohlsson
Exp.tid 09.30 – 11.30 samt 12.45 – 13.30

Handledare:

Doktorander:
MSc. Gustav Cedersjö, Tekn. Lic. Maj Stenmark
Teknologer:
Anders Buhl, Anna Palmqvist Sjövall, Anton Andersson, Cecilia Lindskog, Emil Wihlander, Erik Bjäreholt, Erik Grampp, Filip Stjernström, Fredrik Danebjer, Henrik Olsson, Jakob Hök, Jonas Danebjer, Måns Magnusson, Oscar Sigurdsson, Oskar Berg, Oskar Widmark, Sebastian Hegardt, Stefan Jonsson, Tom Postema, Valthor Halldorsson

Kursmoment — varför?

- **Föreläsningar**: skapa översikt, ge struktur, förklara teori, svara på frågor, motivera varför.
- **Övningar**: bearbeta teorins steg för steg, **grundövningar** för alla, **extraövningar** om du vill/behöver öva mer, **fördjupningsövningar** om du vill gå djupare; **förberedelse inför laborationerna**.
- **Laborationer**: **obligatoriska**, sätta samman teorins delar i ett större program; lösningar redovisas för handledare; gk på alla för att få tenta.
- **Resurstider**: få hjälp med övningar och laborationsförberedelser av handledare, fråga vad du vill.
- **Samarbetsgrupper**: grupplärande genom samarbete, hjälpa varandra.
- **Kontrollskrivning**: **obligatorisk**, diagnostisk, kamraträttad; kan ge samarbetsbonuspoäng till tentan.
- **Individuell projektuppgift**: **obligatorisk**, du visar att du kan skapa ett större program självständigt; redovisas för handledare.
- **Tentamen**: **obligatorisk**, skriftlig, enda hjälpmedel: snabbreferensen. `http://cs.lth.se/pgk/quickref`

Detta är bara början...

Exempel på efterföljande kurser som bygger vidare på denna:

- **Årskurs 1**
  - Programmeringsteknik – fördjupningskurs
  - Utvärdering av programvarusystem
  - Diskreta strukturer
- **Årskurs 2**
  - Objektorienterad modellering och design
  - Programvaruutveckling i grupp
  - Algoritmer, datastrukturer och komplexitet
  - Funktionsprogrammering

Registrering

- Fyll i listan som skickas runt.
- Kryssa i kolumnen **Ska gå** om du ska gå kursen[1][2]
- Kryssa i kolumnen **Kursombud** om du kan tänka dig att vara kursombud under kursens gång:
  - Alla LTH-kurser ska utvärderas under kursens gång och efter kursens slut.
  - Till det behövs kursombud – ungefär **2 D-are** och **2 W-are**.
  - Ni kommer att bli kontaktade av studierådet.

[1] D1:a som redan gått motsvarande högskolekurs? Uppsök studievägledningen

[2] D2:a eller äldre som redan påbörjad EDA016/EDA011/EDA017 el likn.? Övergångsregler: Alla labbar gk: tenta EDA011/017; annars kom och prata på rasten

Förkunskaper

- Förkunskaper $\neq$ Förmåga
- Varken kompetens eller personliga egenskaper är statiska
- ”Programmeringskompetens” är inte *en* enda enkel förmåga utan en komplex sammansättning av flera olika förmågor som **utvecklas** genom hela livet
- Ett innovativt utvecklar**team** behöver många olika kompetenser för att vara framgångsrikt

Förkunskapsenkät

- Om du inte redan gjort det fyll i förkunskapsenkäten **snarast**: `http://cs.lth.se/pgk/survey`
- Dina svar behandlas internt och all redovisad statistik anonymiseras.
- Enkäten ligger till grund för randomiserad gruppindelning i samarbetsgrupper, så att det blir en spridning av förkunskaper inom gruppen.
- Gruppindelnig publiceras här: `http://cs.lth.se/pgk/grupper/`

Samarbetgrupper

- Ni delas in i **samarbetsgrupper** om ca 5 personer baserat på förkunskapsenkäten, så att olika förkunskapsnivåer sammanförs
- Några av laborationerna är mer omfattande **grupplabbar** och kommer att göras i samarbetsgrupperna
- Kontrollskrivningen i halvtid kan ge **samarbetsbonus** (max 5p) som adderas till ordinarie tentans poäng (max 100p) med medelvärdet av gruppmedlemmarnas individuella kontrollskrivningspoäng

  Bonus b för varje person i en grupp med n medlemmar med p_i poäng vardera på kontrollskrivningen:

$$b = \sum_{i=1}^{n} \frac{p_i}{n}$$

Varför studera i samarbetsgrupper?

Huvudsyfte: **Bra lärande!**

- Pedagogisk forskning stödjer tesen att lärandet blir mer djupinriktat om det sker i utbyte med andra
- Ett studiesammanhang med **höga ambitioner** och **respektfull gemenskap** gör att vi **når mycket längre**
- Varför ska du som redan kan mycket aktivt dela med dig av dina kunskaper?
  - Förstå bättre själv genom att förklara för andra
  - Träna din pedagogiska förmåga
  - Förbered dig för ditt kommande yrkesliv som mjukvaruutvecklare

Samarbetskontrakt

Gör ett skriftligt **samarbetskontrakt** med dessa och ev. andra punkter som ni också tycker bör ingå:

1. Återkommande mötestider per vecka
2. Kom i tid till gruppmöten
3. Var väl förberedd genom självstudier inför gruppmöten
4. Hjälp varandra att förstå, men ta inte över och lös allt
5. Ha ett respektfullt bemötande även om ni har olika åsikter
6. Inkludera alla i gemenskapen

Diskutera hur ni ska uppfylla dessa innan alla skriver på.
Ta med samarbetskontraktet och visa för handledare på labb 1.

Om arbetet i samarbetsgruppen inte fungerar ska ni mejla kursansvarig och boka mötestid!

Bestraffa inte frågor!

- Det finns bättre och sämre frågor vad gäller hur mycket man kan lära sig av svaret, men **all undran är en chans** att i dialog utbyta erfarenheter och lärande
- Den som frågar **vill veta** och berättar genom frågan något om nuvarande kunskapsläge
- Den som svarar får chansen att **reflektera** över vad som kan vara svårt och olika vägar till djupare förståelse
- I en hälsosam lärandemiljö är det **helt tryggt** att visa att man ännu inte förstår, att man gjort ”fel”, att man har mer att lära, etc.
- Det är viktigt att våga försöka även om det blir ”fel”: **det är ju då man lär sig!**

Plagiatregler

Läs dessa regler noga och diskutera i samarbetsgrupperna:

- `http://cs.lth.se/utbildning/samarbete-eller-fusk/`
- Föreskrifter angående obligatoriska moment

Ni ska lära er genom **eget arbete** och genom **bra samarbete**. Samarbete gör att man lär sig bättre, men man lär sig inte av att bara kopiera andras lösningar. **Plagiering är förbjuden** och kan medföra **disciplinärende och avstängning**.

En typisk kursvecka

1. Gå på **föreläsningar** på **måndag–tisdag**
2. **Jobba individuellt** med teori, övningar, labbförberedelser på **måndag–torsdag**
3. Kom till **resurstiderna** och få hjälp och tips av handledare och kurskamrater på **onsdag–torsdag**
4. Genomför den obligatoriska **laborationen** på **fredag**
5. **Träffas** i **samarbetsgruppen** och hjälp varandra att förstå mer och fördjupa lärandet, förslagsvis på återkommande tider varje vecka då alla i gruppen kan

Se detaljerna och undantagen i schemat: cs.lth.se/pgk/schema

Laborationer

- **Programmering lär man sig bäst genom att programmera...**
- Labbarna är **individuella** (utom 3) och **obligatoriska**
- Gör **övningarna** och **labbförberedelserna** noga **innan** själva labben – detta är ofta helt nödvändigt för att du ska hinna klart. Dina labbförberedelserna kontrolleras av handledare under labben.
- Är du **sjuk?** Anmäl det **före** labben till `bjorn.regnell@cs.lth.se`, få hjälp på resurstid och redovisa på resurstid (eller labbtid, när handledaren har tid över)
- Hinner du inte med hela labben? Se till att handledaren **noterar din närvaro**, och fortsätt på resurstid och ev. uppsamlingstider.
- Läs noga kapitel noll ”**Anvisningar**” i kompendiet!
- Laborationstiderna är gruppindelade enligt schemat. Du ska gå till den tid och den sal som motsvarar din grupp som visas i TimeEdit. **Gruppindelning** meddelas på hemsidan senast onsdag morgon.

Resurstider

- På resurstiderna får du **hjälp** med **övningar** och labb**förberedelser**.
- Kom till minst en resurstid per vecka, se TimeEdit.
- Handledare gör ibland **genomgångar** för alla under resurstiderna. Tipsa om handledare om vad du finner svårt!
- Du får i mån av plats gå på flera resurstider per vecka. Om det blir fullt i ett rum prioriteras schemagrupper för att minimera krockar:

Tid Lp1	Sal	Grupper med prio
Ons 10-12 v1-7	Falk	09
Ons 10-12 v1-7	Val	10
Ons 13-15 v1-7	Falk	03
Ons 13-15 v1-7	Val	04
Ons 15-17 v1-7	Falk	11
Ons 15-17 v1-7	Val	12
Tor 10-12 v1-7	Falk	01
Tor 10-12 v1-7	Val	02
Tor 13-15 v1-7	Falk	05
Tor 13-15 v1-7	Val	06
Tor 15-17 v1-7	Falk	07
Tor 15-17 v1-7	Val	08

Att lära denna läsvecka w01

Att lära denna läsvecka w01

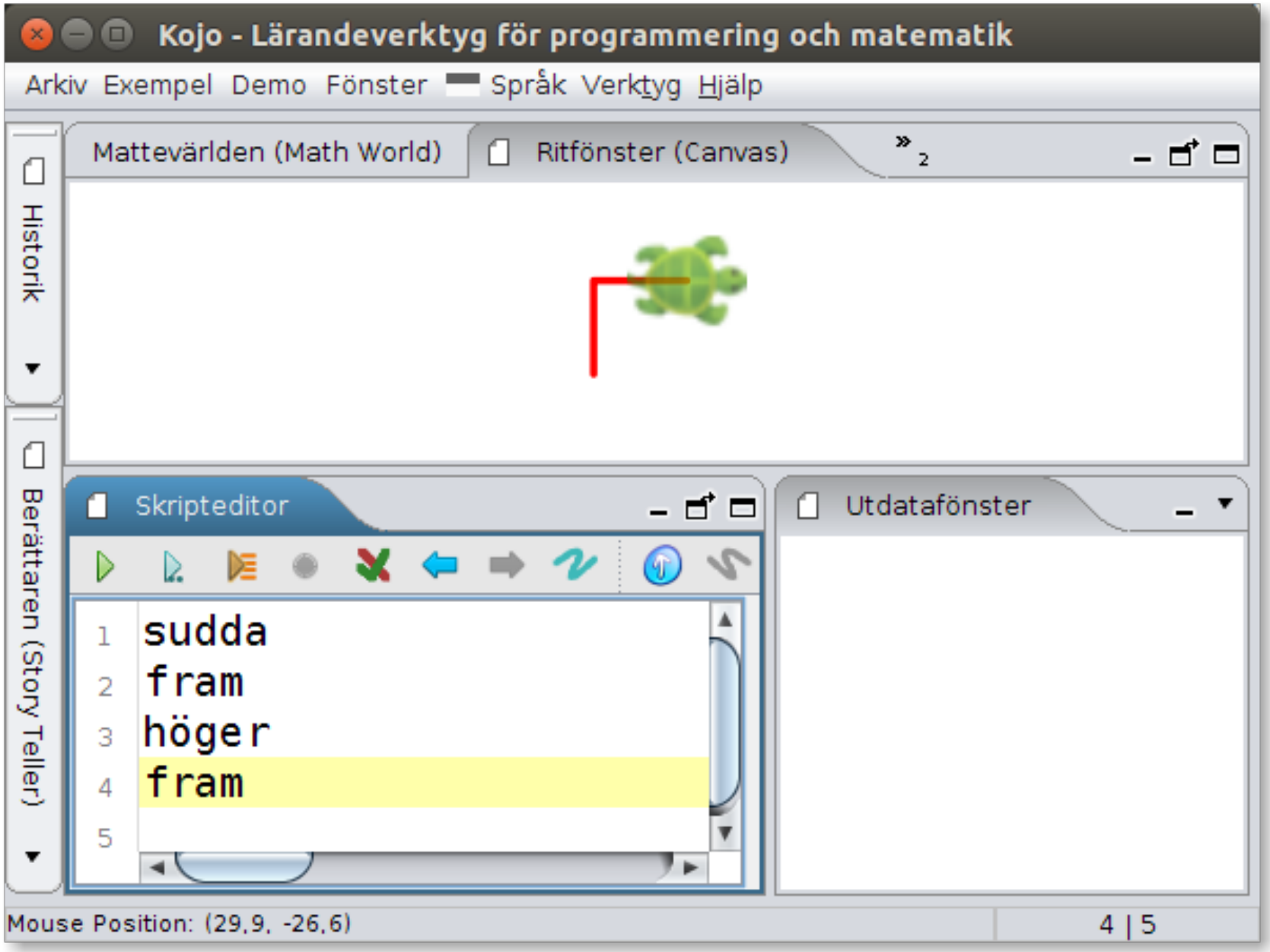
Modul **Introduktion**: Övn **expressions** → Labb **kojo**

- ☐ sekvens
- ☐ alternativ
- ☐ repetition
- ☐ abstraktion
- ☐ programmeringsspråk
- ☐ programmer-ingsparadigmer
- ☐ editera-kompilera-exekvera
- ☐ datorns delar
- ☐ virtuell maskin
- ☐ REPL
- ☐ literal
- ☐ värde
- ☐ uttryck
- ☐ identifierare
- ☐ variabel
- ☐ typ
- ☐ tilldelning
- ☐ namn
- ☐ val
- ☐ var
- ☐ def
- ☐ inbyggda grundtyper
- ☐ Int
- ☐ Long
- ☐ Short
- ☐ Double
- ☐ Float
- ☐ Byte
- ☐ Char
- ☐ String
- ☐ println
- ☐ typen Unit
- ☐ enhetsvärdet ()
- ☐ stränginterpolatorn s
- ☐ if
- ☐ else
- ☐ true
- ☐ false
- ☐ MinValue
- ☐ MaxValue
- ☐ aritmetik
- ☐ slumptal
- ☐ math.random
- ☐ logiska uttryck
- ☐ de Morgans lagar
- ☐ while-sats
- ☐ for-sats

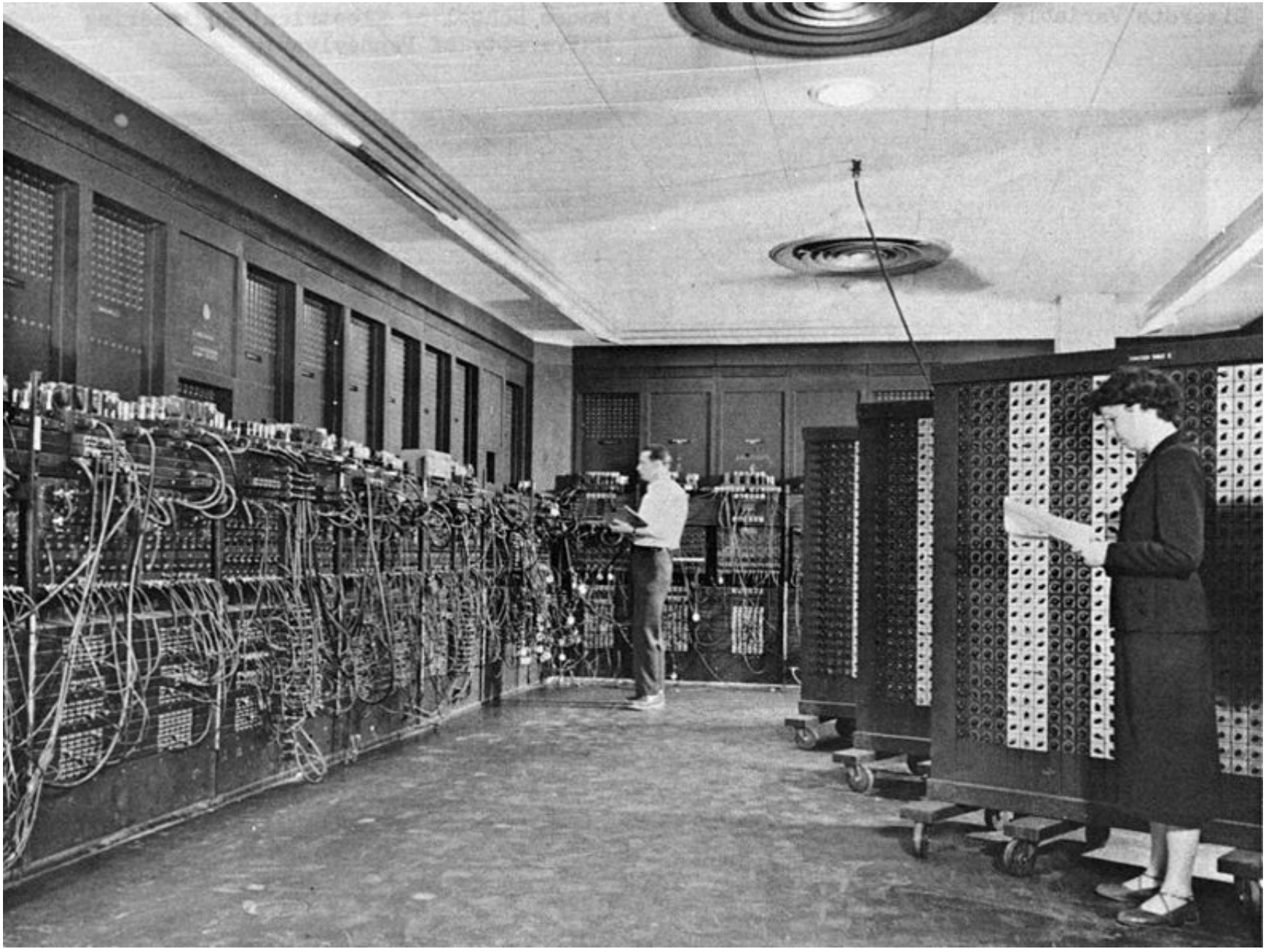
Om programmering

Programming unplugged: Två frivilliga?

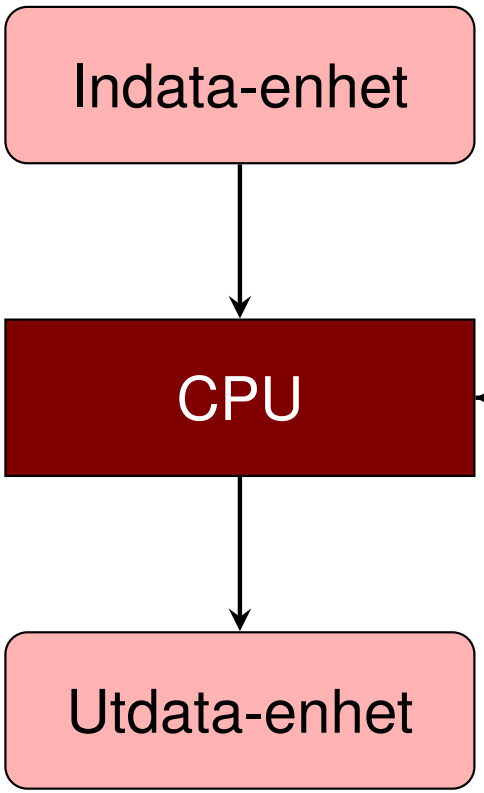
Editera och exekvera ett program



Vad är en dator?



Hur fungerar en dator?



Minne med minnesceller

address	innehåll
0	42
1	13
2	18
3	21
4	55
5	64
6	48
...	...

Minnet innehåller endast **heltal** som representerar **data och instruktioner**.

Vad är programmering?

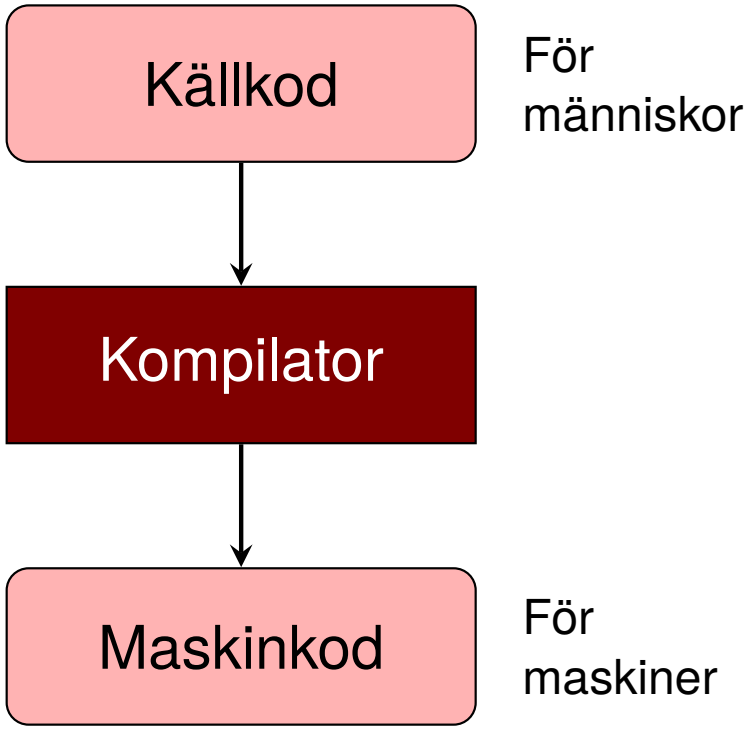
- Programmering innebär att ge instruktioner till en maskin.
- Ett **programmeringsspråk** används av människor för att skriva **källkod** som kan översättas av en **kompilator** till **maskinspråk** som i sin tur **exekveras** av en dator.
- Ada Lovelace skrev det första programmet redan på 1800-talet ämnat för en kugghjulsdator.

- sv.wikipedia.org/wiki/Programmering
- en.wikipedia.org/wiki/Computer_programming
- Ha picknick i Ada Lovelace-parken på Brunnshög!

Vad är en kompilator?

Grace Hopper uppfann första kompilatorn 1952.

en.wikipedia.org/wiki/Grace_Hopper

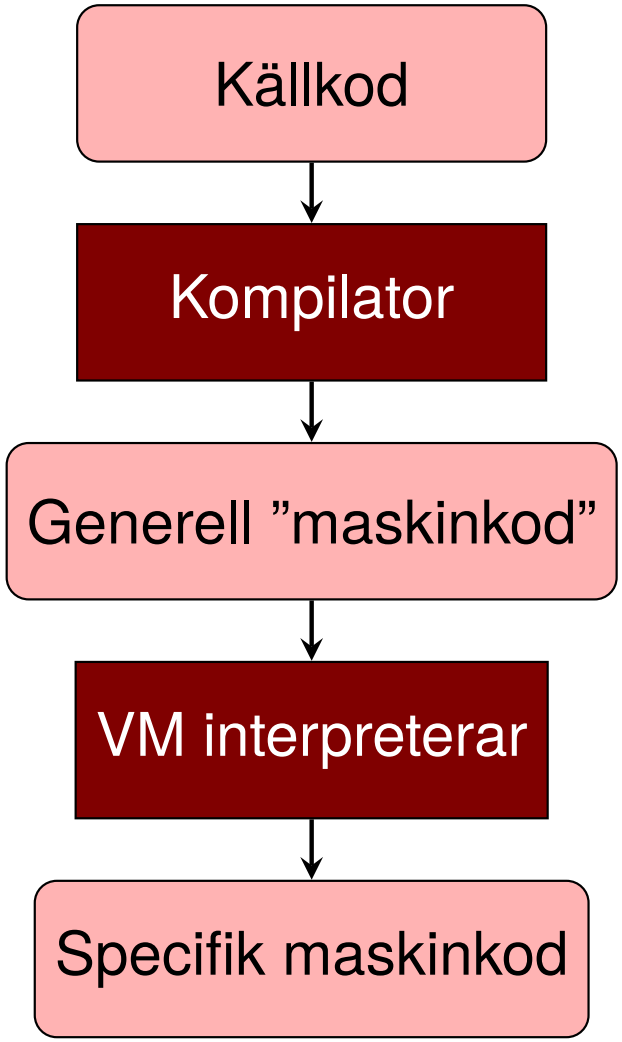


Virtuell maskin (VM) == abstrakt hårdvara

En VM är en ”dator” implementerad i mjukvara som kan tolka en generell ”maskinkod” som **översätts under körning** till den verkliga maskinens kod.

Med en VM blir källkoden **plattformsoberoende** och fungerar på många olika maskiner.

Exempel:
Java Virtual Machine



Vad består ett program av?

- Text som följer entydiga språkregler (gramatik):
  - **Syntax**: textens konkreta utseende
  - **Semantik**: textens betydelse (vad maskinen gör/beräknar)
- **Nyckelord**: ord med speciell betydelse, t.ex. `if`, `else`
- **Deklarationer**: definitioner av nya ord: `def gurka = 42`
- **Satser** är instruktioner som *gör* något: `print("hej")`
- **Uttryck** är instruktioner som beräknar ett *resultat*: `1 + 1`
- **Data** är information som behandlas: t.ex. heltalet 42
- Instruktioner ordnas i kodstrukturer: (SARA)
  - **Sekvens**: ordningen spelar roll för vad som händer
  - **Alternativ**: olika saker händer beroende på uttrycks värde
  - **Repetition**: satser upprepas många gånger
  - **Abstraktion**: nya byggblock skapas för att återanvändas

Exempel på programmeringsspråk

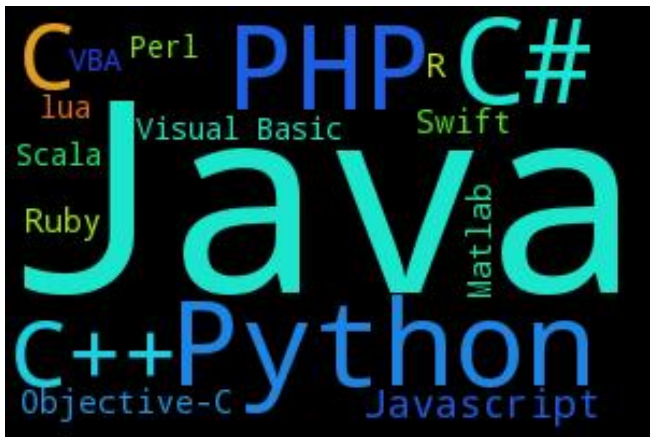
Det finns massor med olika språk och det kommer ständigt nya.

Exempel:

- Java
- C
- C++
- C#
- Python
- JavaScript
- Scala

Topplistor:

- TIOBE Index
- PYPL Index



Hello world

```
scala> println("Hello World!")
Hello World!
```

```scala
// this is Scala

object Hello {
  def main(args: Array[String]): Unit = {
    println("Hejsan scala-appen!")
  }
}
```

```java
// this is Java

public class Hi {
    public static void main(String[] args) {
        System.out.println("Hejsan Java-appen!");
    }
}
```

Utvecklingscykeln

editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; ...

```
upprepa(1000){
  editera
  kompilera
  testa
}
```

Utvecklingsverktyg

- Din verktygskunskap är mycket viktig för din produktivitet.
- Lär dig kortkommandon för vanliga handgrep.
- Verktyg vi använder i kursen:
  - Scala **REPL**: från övn 1
  - **Texteditor** för kod, t.ex `gedit` eller `atom`: från övn 2
  - Kompilera med **`scalac`** och **`javac`**: från övn 2
  - Integrerad utvecklingsmiljö (IDE)
    - **Kojo**: från lab 1
    - **Eclipse+ScalaIDE** eller **IntelliJ IDEA** med Scala-plugin: från lab 3 i vecka 4
  - **jar** för att packa ihop och distribuera klassfiler
  - **javadoc** och **scaladoc** för dokumentation av kodbibliotek
- Andra verktyg som är bra att lära sig:
  - git för versionshantering
  - GitHub för kodlagring – men **inte** av lösningar till labbar!

Att skapa koden som styr världen

I stort sett alla delar av samhället är beroende av programkod:

- kommunikation
- transport
- byggsektorn
- statsförvaltning
- finanssektorn
- media & underhållning
- sjukvård
- övervakning
- integritet
- upphovsrätt
- miljö & energi
- sociala relationer
- utbildning
- ...

Hur blir ditt framtida yrkesliv som systemutvecklare?

- Det är sedan lång tid en skriande brist på utvecklare och bristen blir bara värre och värre...
  CS 2016-08-23
- Störst brist är det på kvinnliga utvecklare:
  DN 2015-04-02
- Global kompetensmarknad
  CS 2015-06-14
  CS 2016-07-14

Utveckling av mjukvara i praktiken

- **Inte bara kodning:** kravbeslut, releaseplanering, design, test, versionshantering, kontinuerlig integration, driftsättning, återkoppling från dagens användare, ekonomi & investering, gissa om morgondagens användare, ...
- **Teamwork:** Inte ensamma hjältar utan autonoma team i decentraliserade organisationer med innovationsuppdrag
- **Snabbhet:** Att koda innebär att hela tiden uppfinna nya ”byggstenar” som ökar organisationens förmåga att snabbt skapa värde med hjälp av mjukvara. Öppen källkod. Skapa kraftfulla API:er.
- **Livslångt lärande:** Lär nytt och dela med dig hela tiden. Exempel på pedagogisk utmaning: hjälp andra förstå och använda ditt API $\Longrightarrow$ *Samarbetskultur*

De enklaste beståndsdelarna: litteraler, uttryck, variabler

Literaler

- Literaler representerar ett fixt **värde** i koden och används för att skapa **data** som programmet ska bearbeta.
- Exempel:

`42`	heltalslitteral
`42.0`	decimaltalslitteral
`'!'`	teckenlitteral, omgärdas med 'enkelfnuttar'
`"hej"`	stränglitteral, omgärdas med "dubbelfnuttar"
`true`	litteral för sanningsvärdet "sant"

- Literaler har en **typ** som avgör vad man kan göra med dem.

Exempel på inbyggda datatyper i Scala

- Alla värden, uttryck och variabler har en **datatyp**, t.ex.:
  - `Int` för heltal
  - `Long` för *extra* stora heltal (tar mer minne)
  - `Double` för decimaltal, så kallade flyttal med flytande decimalpunkt
  - `String` för strängar
- Kompilatorn håller reda på att uttryck kombineras på ett **typsäkert** sätt. Annars blir det kompileringsfel.
- Scala och Java är s.k. **statiskt typade** språk, vilket innebär att all typinformation måste finnas redan vid kompileringsdags (eng. *compile time*)[3].

[3] Andra språk, t.ex. Python och Javascript är dynamiskt typade och där skjuts typkontrollen upp till körningsdags (eng. *run time*)
Vilka är för- och nackdelarna med statisk vs. dynamisk typning?

Grundtyper i Scala

Dessa **grundtyper** (eng. *basic types*) finns inbyggda i Scala:

Svenskt namn	*Engelskt namn*	**Grundtyper**
heltalstyp	integral type	`Byte`, `Short`, `Int`, `Long`, `Char`
flyttalstyp	floating point number types	`Float`, `Double`
numeriska typer	numeric types	heltalstyper och flyttalstyper
strängtyp (teckensekvens)	string type	`String`
sanningsvärdestyp (booelsk typ)	truth value type	`Boolean`

Grundtypernas implementation i JVM

Grundtyp i **Scala**	Antal bitar	Omfång minsta/största värde	**primitiv typ** i **Java** & **JVM**
`Byte`	8	$-2^7 \ldots 2^7 - 1$	`byte`
`Short`	16	$-2^{15} \ldots 2^{15} - 1$	`short`
`Char`	16	$0 \ldots 2^{16} - 1$	`char`
`Int`	32	$-2^{15} \ldots 2^{15} - 1$	`int`
`Long`	64	$-2^{15} \ldots 2^{15} - 1$	`long`
`Float`	32	$\pm\ 3.4028235 \cdot 10^{38}$	`float`
`Double`	64	$\pm\ 1.7976931348623157 \cdot 10^{308}$	`double`

Grundtypen `String` lagras som en *sekvens* av 16-bitars tecken av typen `Char` och kan vara av godtycklig längd (tills minnet tar slut).

Uttryck

- Ett **uttryck** består av en eller flera delar som blir en helhet.
- Delar i ett uttryck kan t.ex. vara:
  literaler (42), operatorer (+), funktioner (sin), ...
- Exempel:
  - Ett enkelt uttryck:
    ```
    42.0
    ```
  - Sammansatta uttryck:
    ```
    40 + 2
    (20 + 1) * 2
    sin(0.5 * Pi)
    "hej" + " på " + "dej"
    ```
- När programmet tolkas sker **evaluering** av uttrycket, vilket ger ett resultat i form av ett **värde** som har en **typ**.

Variabler

- En **variabel** kan tilldelas värdet av ett enkelt eller sammansatt uttryck.
- En variabel har ett **variabelnamn**, vars utformning följer språkets regler för s.k. **identifierare**.
- En ny variabel införs i en **variabeldeklaration** och då den kan ges ett värde, **initialiseras**. Namnet användas som **referens** till värdet.
- Exempel på variabeldeklarationer i Scala, notera **nyckelordet `val`**:

```
val a = 0.5 * Pi
val length = 42 * sin(a)
val exclamationMarks = "!!!"
val greetingSwedish = "Hej på dej" + exclamationMarks
```

- Vid exekveringen av programmet lagras variablernas värden i minnet och deras respektive värde hämtas ur minnet när de **refereras**.
- Variabler som deklareras med **`val`** kan endast tilldelas ett värde **en enda gång**, vid den initialisering som sker vid deklarationen.

Regler för identifierare

- **Enkel** identifierare: t.ex. `gurka2tomat`
  - Börja med bokstav
  - ...följt av bokstäver eller siffror
  - Kan även innehålla understreck
- **Operator**-identifierare, t.ex. `+:`
  - Börjar med ett operatortecken, t.ex. `+ - * / : ? ~ #`
  - Kan följas av fler operatortecken
- **Bokstavlig** identifierare: `` `kan innehålla allt` ``
  - Börjar och slutar med **backticks** `` ` ``
  - Kan innehålla vad som helst (utom backticks)
  - Kan användas för att undvika krockar med reserverade ord: `` `val` ``
- En identifierare får **inte** vara ett **reserverat ord**, se snabbreferensen för alla reserverade ord i Scala & Java.

Att bygga strängar: konkatenering och interpolering

- Man kan **konkatenera** strängar med operatorn +
  `"hej" + " på " + "dej"`
- Efter en sträng kan man konkatenera vilka uttryck som helst som då görs om till en sträng före konkateneringen:

```
val x = 42
val msg = "Dubbla värdet av " + x + " är " + (x * 2) + "."
```

- Man kan i Scala (men inte Java) få hjälp av kompilatorn att övervaka bygget av strängar med **stränginterpolatorn s**:

```
val msg = s"Dubbla värdet av $x är ${x * 2}."
```

Funktioner

Definiera funktioner

Paketet `scala.math`

- math.random

Logik

Alternativ med if-uttryck

Logiska uttryck

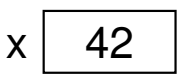
de Morgans lagar

Satser

Tilldelningssatser

Vad är egentligen en variabel?

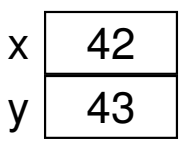
- En variabel har ett **namn** och kan lagra ett **värde** av en viss **typ**
- Variabler måste **deklareras** och då får kompilatorn reda på vilket namnet är och vilken typ av värden som variabeln kan lagra:
  `int x;`
- När variabler deklareras är det oftast bäst att direkt ge dem ett initialvärde:
  `int x = 42;`
- En variabeldeklaration medför att plats i datorns minne reserveras.
  Vi ritar detta såhär:



Dessa deklarationer...

```
int x = 42;
int y = x + 1;
```

... ger detta innehåll någonstans i minnet:

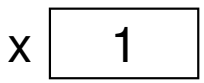


Vad händer egentligen vid en tilldelning?

- Med en **tilldelningssats** kan vi ge en tidigare deklarerad variabel ett nytt värde:

```
x = 1;
```

- Det gamla värdet försvinner för alltid och det nya värdet lagras istället:



- Likhetstecknet används alltså för att *ändra* variablers värden och det är ju *inte* samma sak som matematisk likhet [4]. Vi kan till exempel skriva denna tilldelningssats:

```
x = x + 1;   //Vad händer här?
```

[4] Arv från C, Fortran mfl. I andra språk används t.ex. `x := 42` eller `x <- 42`

Övning: Tilldelningar i sekvens

Rita hur minnet ser ut efter varje rad nedan:

```
1  int u, x, y, z;
2  x = 10;
3  y = 2 * x + 1;
4  z = (y + x) + (y - x);
5  x = x + 1;
```

En variabel som ännu inte initierats har ett odefinierat värde, anges nedan med frågetecken.

	rad 1	rad 2	rad 3	rad 4	rad 5
u	?				
x	?				
y	?				
z	?				

Förkortade tilldelningssatser

- Förkortade tilldelningssatser

Variabler som ändrar värden kan vara knepiga

- Variabler som **förändras** över tid kan vara svåra att resonera kring.
- Många buggar beror på att variabler förändras på felaktiga och oanade sätt.
- Föränderliga värden blir speciellt svåra i kod som körs jämlöpande (parallelt).
- I produktionskod används därför `val` överallt där det går och `var` bara om det verkligen behövs.

Kontrollstrukturer

Används för att kontrollera (förändra) sekvensen.

- if-sats
- while-sats
- for-sats

if-sats

while-sats

for-sats

Procedurer

println

Om veckans övning: `expressions`

Om veckans labb: `kojo`