

# **Programmering, grundkurs**

## Kompendium

EDAA45, Lp1-2, HT 2016  
Datavetenskap, LTH  
Lunds Universitet

<http://cs.lth.se/pgk>

*Editor:* Björn Regnell

*Contributors:* Björn Regnell, ...

*Home:* <https://cs.lth.se/pgk>

*Repo:* <https://github.com/lunduniversity/introprog>

This manuscript is on-going work. Contributions are welcome!

*Contact:* [bjorn.regnell@cs.lth.se](mailto:bjorn.regnell@cs.lth.se)

*LICENCE:* CC BY-SA 4.0

<http://creativecommons.org/licenses/by-sa/4.0/>

Please do *not* distribute your solutions to lab assignments.

Copyright © Computer Science, LTH, Lund University. 2016. Lund. Sweden.

# Framstegsprotokoll

## Genomförda övningar

Till varje laboration hör en övning med uppgifter som utgör förberedelse inför labben. Du behöver minst behärska de grundövningarna för att klara labben inom rimlig tid. Om du känner att du behöver öva mer på grunderna, gör då även extrauppgifterna. Om du vill fördjupa dig, gör fördjupningsuppgifterna som är på mer avancerad nivå. Genom att du kryssar för nedan vilka övningar du har gjort, blir det lättare för handledaren att förstå vilka förkunskaper du har inför labben.

Övning	Grund	Extra	Fördjupning
expressions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
programs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
functions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
data	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
vectors	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
classes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
traits	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
matching	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
matrices	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
sorting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
scalajava	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
threads	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## Godkända obligatoriska moment

För att bli godkänd på laborationsuppgifterna och inlämningsuppgiften måste du lösa deluppgifterna och diskutera dina lösningar med en handledare. Denna diskussion är din möjlighet att få feedback på dina lösningar. Ta vara på den! Se till att handledaren noterar när du blivit godkänd på detta blad, som är ditt kvitto. Spara detta blad tills du fått slutbetyg i kursen.

Namn: .....

Namnteckning: .....

Lab	Datum gk	Handledares namnteckning
kojo	.....	.....
simplewindow	.....	.....
textfiles	.....	.....
cardgame	.....	.....
shapes	.....	.....
turtlerace-team	.....	.....
chords-team	.....	.....
maze	.....	.....
surveydata-team	.....	.....
scalajava-team	.....	.....
life	.....	.....
Inl.Uppg.	.....	.....

### Inlämningsuppgift (välj en)

- ( ) bank
- ( ) mandelbrot
- ( ) draw
- ( ) egendefinerad

*Om egen, ge kort beskrivning:*

# Förord

Programmering är inte bara ett sätt att ta makten över de människoskapade system som är förutsättningen för vårt moderna samhälle. Programmering är också ett kraftfullt verktyg för tanken. Med kunskap i programmeringens grunder kan du påbörja den livslånga läranderesan som det innebär att vara systemutvecklare och abstraktionskonstnär. Programmeringsspråk och utvecklingsverktyg kommer och går, men de grundläggande koncepten bakom *all* mjukvara består: sekvens, alternativ, repetition och abstraktion.

Detta kompendium utgör kursmaterial för en grundkurs i programmering, som syftar till att ge en solid bas för ingenjörsstudenter och andra som vill utveckla system med mjukvara. Materialet omfattar en termins studier på kvartsfart och förutsätter kunskaper motsvarande gymnasienivå i svenska, matematik och engelska.

Kompendiet är framtaget för och av studenter och lärare, och distribueras som öppen källkod. Det får användas fritt så länge erkännande ges och eventuella ändringar publiceras under samma licens som ursprungsmaterialet. På kurshemsidan [cs.lth.se/pgk](http://cs.lth.se/pgk) och i kursrepot [github.com/lunduniversity/introprog](https://github.com/lunduniversity/introprog) finns instruktioner om hur du kan bidra till kursmaterialet.

Läromaterialet fokuserar på lärande genom praktiskt programmeringsarbete och innehåller övningar och laborationer som är organiserade i moduler. Varje modul har ett tema och en teoridel i form av föreläsningsbilder med tillhörande anteckningar.

I kursen använder vi språken Scala och Java för att illustrera grunderna i imperativ och objektorienterad programmering, tillsammans med elementär funktionsprogrammering. Mer avancerad objektorientering och funktionsprogrammering lämnas till efterföljande fördjupningskurser.

Den kanske viktigaste framgångsfaktorn vid studier i programmering är att bejaka din egen upptäckarglädje och experimentlusta. Det fantastiska med programmering är att dina egna intellektuella konstruktioner faktiskt *gör* något som just *du* har bestämt! Ta vara på det och prova dig fram genom att koda egna idéer – det är kul när det funkar men minst lika lärorikt är felsökning, bugggrättande och alla misslyckade försök som efter hårt arbete vänds till lyckade lösningar och/eller bestående lärdomar.

Välkommen i programmeringens fascinerande värld och hjärtligt lycka till med dina studier!

*LTH, Lund 2016*



# Innehåll

<b>Framstegsprotokoll</b>	<b>3</b>
<b>Förord</b>	<b>5</b>
<b>I Om kursen</b>	<b>7</b>
<b>Kursens arkitektur</b>	<b>9</b>
<b>Anvisningar</b>	<b>13</b>
Samarbetsgrupper . . . . .	13
Föreläsningar . . . . .	13
Övningar . . . . .	13
Laborationer . . . . .	13
Resurstider . . . . .	13
Kontrollskrivning . . . . .	13
Tentamen . . . . .	13
<b>Hur bidra till kursmaterialet?</b>	<b>15</b>
<b>II Moduler</b>	<b>17</b>
<b>1 Introduktion</b>	<b>19</b>
1.1 Vad är programmering? . . . . .	20
1.2 Vad är en kompilator? . . . . .	20
1.3 Vad består ett program av? . . . . .	21
1.4 Exempel på programmeringsspråk . . . . .	21
1.5 Varför Scala + Java som förstaspråk? . . . . .	22
1.6 Hello world . . . . .	22
1.7 Utvecklingscykeln . . . . .	23
1.8 Utvecklingsverktyg . . . . .	23
1.9 Övning: expressions . . . . .	24
1.9.1 Grunduppgifter . . . . .	24
1.9.2 Extrauppgifter: öva mer på grunderna . . . . .	31
1.9.3 Fördjupningsuppgifter: avancerad nivå . . . . .	31
1.10 Laboration: kojo . . . . .	33
1.10.1 Obligatoriska uppgifter . . . . .	33

1.10.2	Frivilliga extrauppgifter . . . . .	39
<b>2</b>	<b>Kodstrukturer</b>	<b>41</b>
2.1	Övning: programs . . . . .	42
2.1.1	Grunduppgifter . . . . .	42
2.1.2	Extrauppgifter: öva mer på grunderna . . . . .	43
2.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	43
<b>3</b>	<b>Funktioner, Objekt</b>	<b>45</b>
3.1	Övning: functions . . . . .	46
3.1.1	Grunduppgifter . . . . .	46
3.1.2	Extrauppgifter: öva mer på grunderna . . . . .	46
3.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	46
3.2	Laboration: simplewindow . . . . .	47
3.2.1	Obligatoriska uppgifter . . . . .	47
3.2.2	Frivilliga extrauppgifter . . . . .	47
<b>4</b>	<b>Datastrukturer</b>	<b>49</b>
4.1	Övning: data . . . . .	50
4.1.1	Grunduppgifter . . . . .	50
4.1.2	Extrauppgifter: öva mer på grunderna . . . . .	50
4.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	50
4.2	Laboration: textfiles . . . . .	51
4.2.1	Obligatoriska uppgifter . . . . .	51
4.2.2	Frivilliga extrauppgifter . . . . .	51
<b>5</b>	<b>Vektoralgoritmer</b>	<b>53</b>
5.1	Övning: vectors . . . . .	54
5.1.1	Grunduppgifter . . . . .	54
5.1.2	Extrauppgifter: öva mer på grunderna . . . . .	54
5.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	54
5.2	Laboration: cardgame . . . . .	55
5.2.1	Obligatoriska uppgifter . . . . .	55
5.2.2	Frivilliga extrauppgifter . . . . .	55
<b>6</b>	<b>Klasser, Likhhet</b>	<b>57</b>
6.1	Övning: classes . . . . .	58
6.1.1	Grunduppgifter . . . . .	58
6.1.2	Extrauppgifter: öva mer på grunderna . . . . .	58
6.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	58
6.2	Laboration: shapes . . . . .	59
6.2.1	Obligatoriska uppgifter . . . . .	59
6.2.2	Frivilliga extrauppgifter . . . . .	59
<b>7</b>	<b>Arv, Gränssnitt</b>	<b>61</b>
7.1	Övning: traits . . . . .	62
7.1.1	Grunduppgifter . . . . .	62
7.1.2	Extrauppgifter: öva mer på grunderna . . . . .	62



7.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	62
7.2	Laboration: turtlerace-team . . . . .	63
7.2.1	Obligatoriska uppgifter . . . . .	63
7.2.2	Frivilliga extrauppgifter . . . . .	63
<b>8</b>	<b>Mönster, Undantag</b>	<b>65</b>
8.1	Övning: matching . . . . .	66
8.1.1	Grunduppgifter . . . . .	66
8.1.2	Extrauppgifter: öva mer på grunderna . . . . .	66
8.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	66
8.2	Laboration: chords-team . . . . .	67
8.2.1	Obligatoriska uppgifter . . . . .	67
8.2.2	Frivilliga extrauppgifter . . . . .	67
<b>9</b>	<b>Matriser</b>	<b>69</b>
9.1	Övning: matrices . . . . .	70
9.1.1	Grunduppgifter . . . . .	70
9.1.2	Extrauppgifter: öva mer på grunderna . . . . .	70
9.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	70
9.2	Laboration: maze . . . . .	71
9.2.1	Obligatoriska uppgifter . . . . .	71
9.2.2	Frivilliga extrauppgifter . . . . .	71
<b>10</b>	<b>Sökning, Sortering</b>	<b>73</b>
10.1	Övning: sorting . . . . .	74
10.1.1	Grunduppgifter . . . . .	74
10.1.2	Extrauppgifter: öva mer på grunderna . . . . .	74
10.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	74
10.2	Laboration: surveydata-team . . . . .	75
10.2.1	Obligatoriska uppgifter . . . . .	75
10.2.2	Frivilliga extrauppgifter . . . . .	75
<b>11</b>	<b>Scala och Java</b>	<b>77</b>
11.1	Övning: scalajava . . . . .	78
11.1.1	Grunduppgifter . . . . .	78
11.1.2	Extrauppgifter: öva mer på grunderna . . . . .	78
11.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	78
11.2	Laboration: scalajava-team . . . . .	79
11.2.1	Obligatoriska uppgifter . . . . .	79
11.2.2	Frivilliga extrauppgifter . . . . .	79
<b>12</b>	<b>Trådar, Web, Android</b>	<b>81</b>
12.1	Övning: threads . . . . .	82
12.1.1	Grunduppgifter . . . . .	82
12.1.2	Extrauppgifter: öva mer på grunderna . . . . .	82
12.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	82
12.2	Laboration: life . . . . .	83

12.2.1	Obligatoriska uppgifter . . . . .	83
12.2.2	Frivilliga extrauppgifter . . . . .	83
<b>13</b>	<b>Design</b>	<b>85</b>
<b>14</b>	<b>Tentaträning</b>	<b>87</b>
<b>III</b>	<b>Appendix</b>	<b>89</b>
<b>A</b>	<b>Terminalfönster och kommandoskal</b>	<b>91</b>
A.1	Vad är ett terminalfönster? . . . . .	91
A.2	Några viktiga terminalkommando . . . . .	91
<b>B</b>	<b>Editera</b>	<b>93</b>
B.1	Vad är en editor? . . . . .	93
B.2	Välj editor . . . . .	93
<b>C</b>	<b>Kompilera och exekvera</b>	<b>95</b>
C.1	Vad är en kompilator? . . . . .	95
C.2	Java JDK . . . . .	95
C.2.1	Installera Java JDK . . . . .	95
C.3	Scala . . . . .	95
C.3.1	Installera Scala-kompilatorn . . . . .	95
C.4	Read-Evaluate-Print-Loop (REPL) . . . . .	95
C.4.1	Scala REPL . . . . .	95
<b>D</b>	<b>Dokumentation</b>	<b>97</b>
D.1	Vad gör ett dokumentationsverktyg? . . . . .	97
D.2	scaladoc . . . . .	97
D.3	javadoc . . . . .	97
<b>E</b>	<b>Integrerad utvecklingsmiljö</b>	<b>99</b>
E.1	Vad är en IDE? . . . . .	99
E.2	Kojo . . . . .	99
E.2.1	Installera Kojo . . . . .	99
E.2.2	Använda Kojo . . . . .	99
E.3	Eclipse och ScalaIDE . . . . .	99
E.3.1	Installera Eclipse och ScalaIDE . . . . .	99
E.3.2	Använda Eclipse och ScalaIDE . . . . .	99
<b>F</b>	<b>Byggverktyg</b>	<b>101</b>
F.1	Vad gör ett byggverktyg? . . . . .	101
F.2	Byggverktyget sbt . . . . .	101
F.2.1	Installera sbt . . . . .	101
F.2.2	Använda sbt . . . . .	101

<b>G</b>	<b>Versionshantering och kodlagring</b>	<b>103</b>
G.1	Vad är versionshantering? . . . . .	103
G.2	Versionshanteringsverktyget git . . . . .	103
G.2.1	Installera git . . . . .	103
G.2.2	Använda git . . . . .	103
G.3	Vad är nyttan med en kodlagringsplats? . . . . .	103
G.4	Kodlagringsplatsen GitHub . . . . .	103
G.4.1	Installera klienten för GitHub . . . . .	103
G.4.2	Använda GitHub . . . . .	103
G.5	Kodlagringsplatsen Atlassian BitBucket . . . . .	103
G.5.1	Installera SourceTree . . . . .	103
G.5.2	Använda SourceTree . . . . .	103
<b>H</b>	<b>Nyckelord</b>	<b>105</b>
H.1	Vad är ett nyckelord ord? . . . . .	105
<b>I</b>	<b>Lösningförslag till övningar</b>	<b>107</b>
I.1	expressions . . . . .	108
I.1.1	Grunduppgifter . . . . .	108
I.1.2	Extrauppgifter: öva mer på grunderna . . . . .	108
I.1.3	Fördjupningsuppgifter: avancerad nivå . . . . .	108
I.2	programs . . . . .	109
I.2.1	Grunduppgifter . . . . .	109
I.2.2	Extrauppgifter: öva mer på grunderna . . . . .	109
I.2.3	Fördjupningsuppgifter: avancerad nivå . . . . .	109
I.3	functions . . . . .	110
I.3.1	Grunduppgifter . . . . .	110
I.3.2	Extrauppgifter: öva mer på grunderna . . . . .	110
I.3.3	Fördjupningsuppgifter: avancerad nivå . . . . .	110
I.4	data . . . . .	111
I.4.1	Grunduppgifter . . . . .	111
I.4.2	Extrauppgifter: öva mer på grunderna . . . . .	111
I.4.3	Fördjupningsuppgifter: avancerad nivå . . . . .	111
I.5	vectors . . . . .	112
I.5.1	Grunduppgifter . . . . .	112
I.5.2	Extrauppgifter: öva mer på grunderna . . . . .	112
I.5.3	Fördjupningsuppgifter: avancerad nivå . . . . .	112
I.6	classes . . . . .	113
I.6.1	Grunduppgifter . . . . .	113
I.6.2	Extrauppgifter: öva mer på grunderna . . . . .	113
I.6.3	Fördjupningsuppgifter: avancerad nivå . . . . .	113
I.7	traits . . . . .	114
I.7.1	Grunduppgifter . . . . .	114
I.7.2	Extrauppgifter: öva mer på grunderna . . . . .	114
I.7.3	Fördjupningsuppgifter: avancerad nivå . . . . .	114
I.8	matching . . . . .	115
I.8.1	Grunduppgifter . . . . .	115

I.8.2	Extrauppgifter: öva mer på grunderna . . . . .	115
I.8.3	Fördjupningsuppgifter: avancerad nivå . . . . .	115
I.9	matrices . . . . .	116
I.9.1	Grunduppgifter . . . . .	116
I.9.2	Extrauppgifter: öva mer på grunderna . . . . .	116
I.9.3	Fördjupningsuppgifter: avancerad nivå . . . . .	116
I.10	sorting . . . . .	117
I.10.1	Grunduppgifter . . . . .	117
I.10.2	Extrauppgifter: öva mer på grunderna . . . . .	117
I.10.3	Fördjupningsuppgifter: avancerad nivå . . . . .	117
I.11	scalajava . . . . .	118
I.11.1	Grunduppgifter . . . . .	118
I.11.2	Extrauppgifter: öva mer på grunderna . . . . .	118
I.11.3	Fördjupningsuppgifter: avancerad nivå . . . . .	118
I.12	threads . . . . .	119
I.12.1	Grunduppgifter . . . . .	119
I.12.2	Extrauppgifter: öva mer på grunderna . . . . .	119
I.12.3	Fördjupningsuppgifter: avancerad nivå . . . . .	119

# **Del I**

## **Om kursen**



# Kursens arkitektur

## Veckoöversikt

<i>W</i>	<i>Modul</i>	<i>Övn</i>	<i>Lab</i>
W01	Introduktion	expressions	kojo
W02	Kodstrukturer	programs	–
W03	Funktioner, Objekt	functions	simplewindow
W04	Datastrukturer	data	textfiles
W05	Vektoralgoritmer	vectors	cardgame
W06	Klasser, Likhet	classes	shapes
W07	Arv, Gränssnitt	traits	turtlerace-team
KS	KONTROLLSKRIVN.	–	–
W08	Mönster, Undantag	matching	chords-team
W09	Matriser	matrices	maze
W10	Sökning, Sortering	sorting	surveydata-team
W11	Scala och Java	scalajava	scalajava-team
W12	Trådar, Web, Android	threads	life
W13	Design	Uppsamling	Inl.Uppg.
W14	Tentaträning	Extenta	–
T	TENTAMEN	–	–

Kursen består av ett antal moduler med tillhörande teori, övningar och laborationer. Genom att göra övningarna bearbetar du teorin och förebereder dig inför laborationerna. När du klarat av laborationen i varje modul är du redo att gå vidare till efterkommande modul.

## Vad lär du dig?

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

## 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 föreläsningsanteckningar, övningar & laborationer
- Säljs på KFS  
<http://www.kfsab.se/>

### Rekommenderade böcker

För nybörjare:



För de som redan kodat en del:





## Kursmoment — varför?

- **Föreläsningar:** skapa översikt, ge struktur, förklara teori, svara på frågor, motivera varför
- **Övningar:** bearbeta teorin med avgränsade problem, grundövningar för alla, extraövningar om du behöver öva mer, fördjupningsövningar om du vill gå vidare, **förberedelse** inför laborationerna
- **Laborationer:** lösa programmeringsproblem praktiskt, **obligatoriska** uppgifter; lösningar redovisas för handledare
- **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
- **Inlämningsuppgift:** **obligatorisk**, du visar att du kan skapa ett större program självständigt; redovisas för handledare
- **Tenta:** Skriftlig tentamen utan hjälpmedel, förutom **snabbreferens**.

## 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

## En typisk kursvecka

1. Gå på **föreläsningar** på **måndag-tisdag**
2. Jobba med **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](https://cs.lth.se/pgk/schema)

# Anvisningar

**Samarbetsgrupper**

**Samarbetskontrakt**

**Föreläsningar**

**Övningar**

**Laborationer**

**Resurstider**

**Kontrollskrivning**

**Tentamen**



## **Hur bidra till kursmaterialet?**



# **Del II**

## **Moduler**





# Kapitel 1

## Introduktion

- sekvens
- alternativ
- repetition
- abstraktion
- programmeringsspråk
- programmeringsparadigmer
- editera-kompilera-exekvera
- datorns delar
- virtuell maskin
- värde
- uttryck
- variabel
- typ
- tilldelning
- namn
- val
- var
- def
- if
- else
- true
- false
- MinValue
- MaxValue
- aritmetik
- slump
- math.random
- logiska uttryck
- de Morgans lagar
- while-sats
- for-sats

## 1.1 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.
- Ha picknick i Ada Lovelace-parken på Brunshög!



- [sv.wikipedia.org/wiki/Programmering](https://sv.wikipedia.org/wiki/Programmering)
- [en.wikipedia.org/wiki/Computer\\_programming](https://en.wikipedia.org/wiki/Computer_programming)
- [kartor.lund.se/wiki/lundanamn/index.php/Ada\\_Lovelace-parken](https://kartor.lund.se/wiki/lundanamn/index.php/Ada_Lovelace-parken)

## 1.2 Vad är en kompilator?



Grace Hopper uppfann första kompilatorn 1952.

[en.wikipedia.org/wiki/Grace\\_Hopper](https://en.wikipedia.org/wiki/Grace_Hopper)



## 1.3 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**
- **Deklaration**: 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

## 1.4 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](#)



## 1.5 Varför Scala + Java som förstaspråk?

- Varför Scala?
  - Enkel och enhetlig syntax => lätt att skriva
  - Enkel och enhetlig semantik => lätt att fatta
  - Kombinerar flera angreppssätt => 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 plattformar
  - Effektivitet: avancerad & mogen teknik ger snabba program
- Java och Scala fungerar utmärkt tillsammans
- Illustrera likheter och skillnader mellan olika språk  
=> Djupare lärande

## 1.6 Hello world

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

```
// this is Scala
```

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

```
// this is Java
```

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

## 1.7 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; ...

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

## 1.8 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: från övn 2
  - Kompilera med **scalac** och **javac**: från övn 2
  - Integrerad utvecklingsmiljö (IDE)
    - \* **Kojo**: från lab 1
    - \* **Eclipse** med plugin **ScalaIDE**: från lab 3
  - **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!

## 1.9 Övning: expressions

### Mål

- Förstå vad som händer när satser exekveras och uttryck evalueras.
- Förstå sekvens, alternativ och repetition.
- Känna till literalerna för enkla värden, deras typer och omfång.
- Kunna deklarerar och använda variabler och tilldelning, samt kunna rita bilder av minnessituationen då variablers värden förändras.
- Förstå skillnaden mellan olika numeriska typer, kunna omvandla mellan dessa och vara medveten om noggrannhetsproblem som kan uppstå.
- Förstå booelska uttryck och värdena **true** och **false**, samt kunna förenkla booelska uttryck.
- Förstå skillnaden mellan heltalsdivision och flyttalsdivision, samt användning av rest vid heltalsdivision.
- Förstå precedensregler och användning av parenteser i uttryck.
- Kunna använda **if**-satser och **if**-uttryck.
- Kunna använda **for**-satser och **while**-satser.
- Kunna använda `math.random` för att generera slumpantal i olika intervall.

### Förberedelser

- Studera teorin i kapitel 1.
- Du behöver en dator med Scala installerad; se appendix C.

#### 1.9.1 Grunduppgifter

**Uppgift 1.** Starta Scala REPL (eng. *Read-Evaluate-Print-Loop*) och skriv satsen `println("hejsan REPL")` och tryck på *Enter*.

```
> scala
Welcome to Scala version 2.11.7 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8).
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("hejsan REPL")
```

- a) Vad händer?
- b) Skriv samma sats igen men "glöm bort" att skriva högerparentesen innan du trycker på *Enter*. Vad händer?
- c) Evaluera uttrycket `"gurka" + "tomat"` i REPL. Vad har uttrycket för värde och typ? Vilken siffra står efter ordet `res` i variabeln som lagrar resultatet?

```
scala> "gurka" + "tomat"
```

- d) Evaluera uttrycket `res0 * 42` men byt ut `0`:an mot siffran efter `res` i utskriften från förra evalueringen. Vad har uttrycket för värde och typ?

```
scala> res2 * 42
```

 **Uppgift 2.** Vad är en *literal*?

**Uppgift 3.** Vilken typ har följande literaler?

- a) 42
- b) 42L
- c) '\*'
- d) "\*"
- e) 42.0
- f) 42D
- g) 42d
- h) 42F
- i) 42f
- j) **true**
- k) **false**

 **Uppgift 4.** Vad gör dessa satser? Till vad används klammer och semikolon?

```
scala> def p = { print("hej"); print("san"); println(42); println("gurka") }  
scala> p;p;p;p
```

 **Uppgift 5.** Satser versus uttryck.

- a) Vad är det för skillnad på en sats och ett uttryck?
- b) Ge exempel på satser som inte är uttryck?
- c) Förklara vad som händer för varje evaluerad rad:

```
1 scala> def värdeSaknas = ()  
2 scala> värdeSaknas  
3 scala> värdeSaknas.toString  
4 scala> println(värdeSaknas)  
5 scala> println(println("hej"))
```

- d) Vilken typ har literalen ()?
- e) Vilken returtyp har println?

**Uppgift 6.** Vilken typ och vilket värde har följande uttryck?

- a) 1 + 41
- b) 1.0 + 41
- c) 42.toDouble
- d) (41 + 1).toDouble
- e) 1.042e42
- f) 42E6.toLong
- g) "gurk" + 'a'
- h) 'A'

- i) `'A'.toInt`
- j) `'0'.toInt`
- k) `'1'.toInt`
- l) `'9'.toInt`
- m) `('A' + '0').toChar`
- n) `"*!%#".charAt(0)`

**Uppgift 7.** *De fyra räknesätten.* Vilket värde och vilken typ har följande uttryck?

- a) `42 * 2`
- b) `42.0 / 2`
- c) `42 - 0.2`
- d) `42L + 2d`

**Uppgift 8.** *Precedensregler.* Evalueringsordningen kan styras med parenteser. Vilket värde och vilken typ har följande uttryck?

- a) `42 + 2 * 2`
- b) `(42 + 2) * 2`
- c) `(-(2 - 42)) / (1 + 1 + 1).toDouble`
- d) `((-(2 - 42)) / (1 + 1 + 1).toDouble).toInt`

**Uppgift 9.** *Heltalsdivision.* Vilket värde och vilken typ har följande uttryck?

- a) `42 / 2`
- b) `42 / 4`
- c) `42.0 / 4`
- d) `1 / 4`
- e) `1 % 4`
- f) `2 % 42`
- g) `42 % 2`

**Uppgift 10.** *Heltalsomfång.* För var och en av heltalstyperna i deluppgifterna nedan: undersök i REPL med operationen `MaxValue` resp. `MinValue`, till exempel `Int.MaxValue` vad som är största och minsta värde.

- a) `Byte`
- b) `Short`
- c) `Int`
- d) `Long`

**Uppgift 11.** Klassen `java.lang.Math` och paketobjektet `scala.math`.

```
scala> java.lang.Math. //tryck TAB
scala> scala.math.     //tryck TAB
```



- a) Undersök genom att trycka på Tab-tangenten, vilka funktioner som finns i Math och math. Vad heter konstanten  $\pi$  i java.lang.Math respektive scala.math?
- b) Undersök dokumentationen för klassen java.lang.Math här:  
<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>  
Vad gör java.lang.Math.hypot?
- c) Undersök dokumentationen för paketobjektet scala.math här:  
<http://www.scala-lang.org/api/current/#scala.math.package>  
Ge exempel på någon funktion i java.lang.Math som inte finns i scala.math.


**Uppgift 12.** Vad händer här? Notera undantag (eng. *exceptions*) och nogrannhetsproblem.

- a) Int.MaxValue + 1
- b) 1 / 0
- c) 1E8 + 1E-8
- d) 1E9 + 1E-9
- e) math.pow(math.hypot(3,6), 2)
- f) 1.0 / 0
- g) (1.0 / 0).toInt
- h) math.sqrt(-1)
- i) math.sqrt(Double.NaN)
- j) **throw new** Exception("PANG!!!")

**Uppgift 13.** Booleska uttryck. Vilket värde och vilken typ har följande uttryck?

- a) **true** && **true**
- b) **false** && **true**
- c) **true** && **false**
- d) **false** && **false**
- e) **true** || **true**
- f) **false** || **true**
- g) **true** || **false**
- h) **false** || **false**
- i) 42 == 42
- j) 42 != 42
- k) 42.0001 == 42
- l) 42.000000000000000001 == 42
- m) 42.0001 > 42
- n) 42.000000000000000001 > 42
- o) 42.0001 >= 42
- p) 42.000000000000000001 <= 42
- q) **true** == **true**

- r) `true != true`
- s) `true > false`
- t) `true < false`
- u) `'A' == 65`
- v) `'S' != 66`

**Uppgift 14.** *Variabler och tilldelning.* Rita en ny bild av datorns minne efter varje evaluerad rad nedan. Bilderna ska visa variablers namn, typ och värde. 


```
1 scala> var a = 42
2 scala> var b = a + 1
3 scala> var c = (a + b) * 2.0
4 scala> b = 0
5 scala> a = 0
6 scala> c = c + 1
```

Efter första raden ser minnessituationen ut så här:

a: Int

**Uppgift 15.** *Deklarationer: `var`, `val`, `def`.* Evaluera varje rad nedan i tur och ordning i Scala REPL.

```
1 scala> var x = 42
2 scala> x + 1
3 scala> x
4 scala> x = x + 1
5 scala> x
6 scala> x == x + 1
7 scala> val y = 42
8 scala> y = y + 1
9 scala> var z = {println("gurka"); 42}
10 scala> def w = {println("gurka"); 42}
11 scala> z
12 scala> z
13 scala> z = z + 1
14 scala> w
15 scala> w
16 scala> w = w + 1
```

- a) För varje rad ovan: förklara för varje rad vad som händer.
- b) Vilka rader ger kompileringsfel och i så fall vilket och varför?
- c) Vad är det för skillnad på `var`, `val` och `def`? 

**Uppgift 16.** *if-sats.* För varje rad nedan; förklara vad som händer.

```
scala> if (true) println("sant") else println("falskt")
scala> if (false) println("sant") else println("falskt")
scala> if (!true) println("sant") else println("falskt")
scala> if (!false) println("sant") else println("falskt")
scala> def kasta = if (math.random > 0.5) println("krona") else println("klave")
scala> kasta; kasta; kasta
```

**Uppgift 17. if-uttryck.** Följande variabler är deklarerade med nedan initialvärden:

```
scala> var grönsak = "gurka"
scala> var frukt = "banan"
```

Vad har följande uttryck för värden och typ?

- a) **if** (grönsak == "tomat") "gott" **else** "inte gott"
- b) **if** (frukt == "banan") "gott" **else** "inte gott"
- c) **if** (frukt.size == grönsak.size) "lika stora" **else** "olika stora"

**Uppgift 18. for-sats.**

- a) Vad ger nedan **for**-satser för utskrift?

```
scala> for (i <- 1 to 10) print(i + ", ")
scala> for (i <- 1 until 10) print(i + ", ")
scala> for (i <- 1 to 5) print((i * 2) + ", ")
scala> for (i <- 1 to 92 by 10) print(i + ", ")
scala> for (i <- 10 to 1 by -1) print(i + ", ")
```

- b) Skriv en **for**-sats som ger följande utskrift:

```
A1, A4, A7, A10, A13, A16, A19, A22, A25, A28, A31, A34, A37, A40, A43,
```

**Uppgift 19. Repetition med foreach.**

- a) Vad ger nedan satser för utskrifter?

```
scala> (9 to 19).foreach{i => print(i + ", ")}
scala> (1 until 20).foreach{i => print(i + ", ")}
scala> (0 to 33 by 3).foreach{i => print(i + ", ")}

```

- b) Använd foreach och skriv ut följande:

```
B33, B30, B27, B24, B21, B18, B15, B12, B9, B6, B3, B0,
```

**Uppgift 20. while-sats.**

- a) Vad ger nedan satser för utskrifter?

```
scala> var i = 0
scala> while (i < 10) { println(i); i = i + 1 }
scala> var j = 0; while (j <= 10) { println(j); j = j + 2 }; println(j)
```

- b) Skriv en **while**-sats som ger följande utskrift. Använd en variabel k som initialiseras till 1.




```
A1, A4, A7, A10, A13, A16, A19, A22, A25, A28, A31, A34, A37, A40, A43,
```



- c) Vilken av **for**, **while** och foreach är kortast att skriva om man vill repetera mer än en sats 100 gånger? Vilken tycker du är lättast att läsa?

**Uppgift 21.** *Slumptal.* Undersök vad dokumentationen säger om funktionen `scala.math.random`:

<http://www.scala-lang.org/api/current/#scala.math.package>

- a) Vilken typ har värdet som returneras av funktionen `random`? 
- b) Vilket är det minsta respektive största värde som kan returneras? 
- c) Är `random` en *äkta* funktion (eng. *pure function*) i matematisk mening? 
- d) Anropa funktionen `math.random` upprepade gånger och notera vad som händer. Använd pil-upp-tangenten.

```
scala> math.random
```

- e) Vad händer? Använd *pil-upp* och kör nedan **for**-sats flera gånger. Förklara vad som sker.

```
scala> for (i <- 1 to 10) println(math.random)
```

- f) Skriv en **for**-sats som skriver ut 100 slumpmässiga heltal från 0 till och med 9 på var sin rad.

```
scala> for (i <- 1 to 100) println(???)
```

- g) Skriv en **for**-sats som skriver ut 100 slumpmässiga heltal från 1 till och med 6 på samm rad.

```
scala> for (i <- 1 to 100) print(???)
```


- h) Använd *pil-upp* och kör nedan **while**-sats flera gånger. Förklara vad som sker.

```
scala> while (math.random > 0.2) { println("gurka") }
```

- i) Ändra i **while**-satsen ovan så att sannolikheten ökar att riktigt många strängar ska skrivas ut.

- j) Förklara vad som händer nedan.

```
scala> var slumptal = math.random  
scala> while (slumptal > 0.2) { println(slumptal); slumptal = math.random }
```

**Uppgift 22.** *Logik och De Morgans Lagar.* Förenkla följande uttryck. Antag att poäng och highscore är heltalsvariabler medan klar är av typen Boolean. 

- a) `poäng > 100 && poäng > 1000`
- b) `poäng > 100 || poäng > 1000`
- c) `!(poäng > highscore)`
- d) `!(poäng > 0 && poäng < highscore)`
- e) `!(poäng < 0 || poäng > highscore)`
- f) `klar == true`
- g) `klar == false`

## 1.9.2 Extrauppgifter: öva mer på grunderna

### Uppgift 23. *Slumptal.*

a) Ersätt ??? nedan med literaler så att `tärning` returnerar ett slumpmässigt heltal mellan 1 och 6.

```
scala> def tärning = (math.random * ??? + ???).toInt
```

b) Ersätt ??? med literaler så att `rnd` blir ett decimaltal med max en decimal mellan 0.0 och 1.0.

```
scala> def rnd = math.round(math.random * ???) / ???
```

c) Vad blir det för skillnad om `math.round` ersätts med `math.floor` ovan? (Se dokumentationen av `java.lang.Math.round` och `java.lang.Math.floor`.)

## 1.9.3 Fördjupningsuppgifter: avancerad nivå

**Uppgift 24.** `Integer.toBinaryString`, `Integer.toHexString`

**Uppgift 25.** Typannoteringar.

**Uppgift 26.** `0x2a`

**Uppgift 27.** `i += 1; i *= 1; i /= 2`

**Uppgift 28.** `BigInt`, `BigDecimal`

**Uppgift 29.** Vad händer här?

```
scala> Math.multiplyExact(2, 42)
scala> Math.multiplyExact(Int.MaxValue, Int.MaxValue)
```

**Uppgift 30.** Sök reda på dokumentationen för funktionen `multiplyExact` i javadoc för klassen `java.lang.Math` i JDK 8.

**Uppgift 31.** Sök i javadoc för `Math` efter förekomster av texten *"throwing an exception if the result overflows"*. Vilka fler funktioner finns i `java.lang.Math` som hjälper en att upptäcka om det blir overflow?

**Uppgift 32.** Använda Scala REPL för att undersöka konstanterna nedan. Vilket av dessa värden är negativt? Vad kan man ha för praktisk nytta av dessa värden i ett program som gör flyttalsberäkningar?

- a) `java.lang.Double.MIN_VALUE`
- b) `scala.Double.MinValue`
- c) `scala.Double.MinPositiveValue`

**Uppgift 33.** För typerna Byte, Short, Char, Int, Long, Float, Double: Undersök hur många bitar som behövs för att representera varje typs omfång?

*Tips:* Några användbara uttryck:

```
Integer.toBinaryString(Int.MaxValue + 1).size
```

```
Integer.toBinaryString((math.pow(2,16) - 1).toInt).size
```

$1 + \text{math.log}(\text{Long.MaxValue})/\text{math.log}(2)$  Se även språkspecifikationen för Scala, kapitlet om heltalslitteraler:

<http://www.scala-lang.org/files/archive/spec/2.11/01-lexical-syntax.html#integer-literals>

a) Undersök källkoden för paketobjektet `scala.math` här:

<https://github.com/scala/scala/blob/v2.11.7/src/library/scala/math/package.scala>

Hur många olika överlagrade varianter av funktionen `abs` finns det och för vilka parametertyper är den definierad?

## 1.10 Laboration: kojo

### Mål

- Kunna kombinera principerna sekvens, alternativ, repetition, och abstraktion i skapandet av egna program om minst 20 rader kod.
- Kunna förklara vad ett program gör i termer av sekvens, alternativ, repetition, och abstraktion.
- Kunna tillämpa principerna sekvens, alternativ, repetition, och abstraktion i enkla algoritmer.
- Kunna formatera egna program så att de blir lätta att läsa och förstå.
- Kunna förklara vad en variabel är och kunna skriva deklARATIONER och göra tilldelningar.
- Kunna genomföra upprepade varv i cykeln *editera-exekvera-felsöka/förbättra* för att succesivt bygga upp allt mer utvecklade program.

### Förberedelser

- Gör övning expressions i kapitel 1.9.
- Läs igenom "Kojo - An Introduction" (25 sidor) som du kan ladda ner i pdf här: <http://www.kogics.net/kojo-ebooks>
- Du behöver en dator med Kojo installerad, se appendix E.2.

### 1.10.1 Obligatoriska uppgifter

#### Uppgift 1. Sekvens.

a) Starta Kojo. Om du inte redan har svenska menyer: välj svenska i språkmenyn och starta om Kojo. Skriv in nedan program och tryck på den *gröna* play-knappen. Du hittar en lista med några fler funktioner på svenska och engelska i appendix E.2.

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

b) Prova att ändra på ordningen mellan satserna och använd den *gula* play-knappen (programspårning) för att studera vad som händer. Klicka på satser i ditt program och på rutor i programspårningen och se vad som händer.

c) Prova satser i sekvens på flera rader, respektive på samma rad med semikolon emellan. Hur vill du gruppera dina satser så att de lätta för en människa att läsa?



d) Vad händer om du *inte* börjar programmet med sudda och kör det upprepade gånger? Varför är det bra börja programmet med sudda?

e) Rita en kvadrat som i bilden nedan.



f) Rita en trappa som i bilden nedan.



g) Rita och mät.

- Börja ditt program med dessa satser:  
sudda; axesOn; gridOn; sakta(0); osynlig
- Rita sedan en kvadrat som har 444 längdenheter i omkrets.
- Ta fram linjalen med höger-klick i ritfönstret och mät så exakt du kan hur lång diagonalen i kvadraten är. Skriv ner resultatet.
- Kontrollera med hjälp av `math.hypot` och `println` vad det exakta svaret är. Skriv ner svaret med 3 decimalers noggrannhet.
- *Tips:* Du kan zooma med mushjulet om du håller nere Ctrl-knappen. Du kan flytta linjalen om du klick-drar på linjalens skalstreck. Du kan vrida linjalen om du klickar på skalstrecken och håller nere Shift-tangenten.

h) Rita en triangel med sidan 300 längdenheter genom att ge lämpliga argument till fram och höger. Vinklar anges i grader och ett helt varv motsvarar 360 grader.

i) Visa dina resultat för en handledare och diskutera hur uppgifterna ovan ✓ 👁 illustrerar principen om sekvens.

### Uppgift 2. Repetition.

- Rita en kvadrat igen, men nu med hjälp av proceduren `upprepa(n){ ??? }` där du ersätter `n` med antalet repetitioner och `???` med de satser som ska repeteras.
- Kör ditt program med den gula play-knappen. Studera hur repetitionen påverkar exekveringssekvensen. Vid vilka punkter i programmet sker ett "hopp" i sekvensen i stället för att efterföljande sats att exekveras? Använd lämpligt argument till `sakta` för att du ska hinna studera exekveringen.
- Anropa proceduren `sakta(???)` med lämplig parameter och gör så att sköldpaddan går totalt 20 varv i kvadraten på ungefär 2 sekunder. *Tips:* Du kan köra ditt program med `Ctrl+Enter` i stället för att trycka på den gröna



play-knappen. Anropa sakta i början av ditt program men *efter* sudda. (Vad händer om du anropar sakta före sudda?)

d) Om du anropar sakta(0), hur många kvadratvarv hinner sköldpaddan rita på en sekund? Använd nedan program för att ta reda på ungefärligt antal varv per sekund.

```
sudda; sakta(0)
val t1 = System.currentTimeMillis
upprepa(800*4){fram;höger}
val t2 = System.currentTimeMillis
println("Det tog " + (t2 - t1) + " millisekunder")
```

e) Rita en kvadrat igen, men nu med hjälp av en **while**-sats och en loop-variabel.

```
var i = 0
while (???) {fram; höger; i = ???}
```

f) Rita en kvadrat igen, men nu med hjälp av en **for**-sats.

```
for (i <- 1 to ???) {???
```

g) Rita en kvadrat igen, men nu med hjälp av foreach.

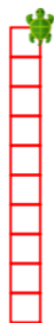
```
(1 to ???).foreach{i => ???}
```

- ✓ 👁️ ✎ h) Vad är fördelar och nackdelar med de olika sätten att loopa: upprepa, **while**, **for**, respektive foreach? Diskutera dina svar med en handledare.

### Uppgift 3. Abstraktion.

a) Använd proceduren kvadrat nedan och proceduren hoppa(???) för att rita en stapel med 10 kvadrater enligt bilden.

```
def kvadrat = for (i <- 1 to 4) {fram; höger}
```



b) Kör ditt program med den *gula* play-knappen. Studera hur anrop av abstraktionen påverkar exekveringssekvensen. Vid vilka punkter i programmet

sker ett ”hopp” i sekvensen i stället för att efterföljande sats att exekveras? Använd lämpligt argument till sakta för att du ska hinna studera exekveringen.

c) Rita samma bild med 10 staplade kvadrater som ovan, men nu *utan* att använda abstraktionen kvadrat – använd i stället en nästlad repetition. Vilket av de två sätten (med och utan abstraktion) är lättast att läsa?

*Tips:* Varje gång du trycker på någon av play-knapparna, sparas ditt program. Du kan se dina sparade program om du klickar på *Historik*-fliken. Du kan också stega bakåt och framåt i historiken med de blå pilarna bredvid play-knapparna.

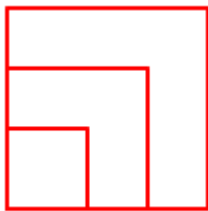
d) Skapa en abstraktion `def stapel = ???` som kör din kod för att rita stapeln.

e) Ge proceduren `stapel` en parameter `n` som styr hur många kvadrater som ritas.

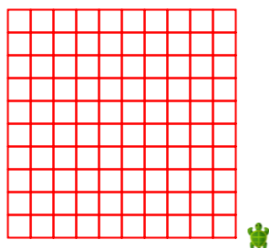
```
def kvadrat = ???
def stapel(n: Int) = ???

sudda; sakta(100)
stapel(42)
```

f) Ge abstraktionen `kvadrat` en parameter `sida: Double` som anger hur stor kvadraten blir. Rita flera kvadrater i likhet med bilden nedan.



g) Rita nedan bild med hjälp av abstraktionen `stapel`. Det är totalt 100 kvadrater och varje kvadrat har sidan 25. *Tips:* Med ett negativt argument till proceduren `hoppa` kan du få sköldpaddan att hoppa baklänges utan att rita, t.ex. `hoppa(-10*25)`



h) Skapa en abstraktion `rutnät` med lämpliga parametrar som gör att man kan rita rutnät med olika stora kvadrater och olika många kvadrater i både x- och y-led.

i) Se över ditt program i föregående uppgift och säkerställ att det är lättläst ✓ 👁 och följer en struktur som börjar med alla definitioner i logisk ordning och därefter fortsätter med huvudprogrammet. Diskutera ditt program med en

händelare. Vad har du gjort för att programmet ska vara lättläst?

#### Uppgift 4. Variabel.

#### Uppgift 5. Alternativ.

a) Kör programmet nedan. Förklara vad som händer. Använd den gula play-knappen för att studera exekveringen.

```
sudda; sakta(5000)

def move(key: Int): Unit = {
  println("key: " + key)
  if (key == 87) fram(10)
  else if (key == 83) fram(-10)
}

move(87); move('W'); move('W')
move(83); move('S'); move('S'); move('S')
```

b) Kör programmet nedan. Notera activateCanvas för att du ska slippa klicka i ritfönstret innan du kan styra paddan. Lägg till kod i move som gör att tangenten A ger en vridning moturs med 5 grader medan tangenten D ger en vridning medurs 5 grader.

```
sudda; sakta(0); activateCanvas

def move(key: Int): Unit = {
  println("key: " + key)
  if (key == 'W') fram(10)
  else if (key == 'S') fram(-10)
}

onKeyPress(move)
```

c) Lägg till nedan kod i början av programmet och gör så att när man trycker på tangenten G så sätter man omväxlande på och av rutnätet.

```
var isGridOn = false

def toggleGrid =
  if (isGridOn) {
    gridOff
    isGridOn = false
  } else {
    gridOn
    isGridOn = true
  }
```

d) Gör så att när man trycker på tangenten X så sätter man omväxlande på och av koordinataxlarna. Använd en variabel `isAxesOn` och definiera en abstraktion `toggleAxes` som anropar `axesOn` och `axesOff` på liknande sätt som i föregående uppgift.

### Uppgift 6. Tidmätning. Hur snabb är din dator?

a) Skriv in koden nedan i Kojos editor och kör med den gröna play-knappen. Hur långt tid tar det för din dator att räkna till 4.4 miljarder?

```
object timer {
  def now: Long = System.currentTimeMillis
  var saved: Long = now
  def elapsedMillis: Long = now - saved
  def elapsedSeconds: Double = elapsedMillis / 1000.0
  def reset: Unit = { saved = now }
}

// HUVUDPROGRAM:
timer.reset
var i = 0L
while (i < 4400000000L) { i += 1 }
val t = timer.elapsedSeconds
println("Räknade till " + i + " på " + t + " sekunder.")
```

b) Om du kör på en Linux-maskin: Kör nedan Linux-kommando upprepade gånger i ett terminalfönster. Med hur många MHz kör din dators klocka för tillfället? Hur förhåller sig klockfrekvensen till antalet runder i while-loopen i föregående uppgift? (Det kan hända att din dator kan variera centralprocessorns klockfrekvens. Prova både medan du kör tidmätningen i Kojo och då din dator "vilar".)

```
> lscpu | grep MHz
```

c) Ändra i koden i uppgift a) så att **while**-loopen bara kör 5 gånger. Kör programmet med den *gula* play-knappen. Scrolla i programspårningen och förklara vad som händer. Klicka på CALL-rutorna och se vilken rad som markeras i ditt program.

d) Lägg till koden nedan i ditt program och försök ta reda på ungefär hur långt din dator hinner räkna till på en sekund för Long- respektive Int-variabler. Använd den gröna play-knappen.

```
def timeLong(n: Long): Double = {
  timer.reset
  var i = 0L
  while (i < n) { i += 1 }
  timer.elapsedSeconds
}
```

```
def timeInt(n: Int): Double = {
  timer.reset
  var i = 0
  while (i < n) { i += 1 }
  timer.elapsedSeconds
}

def show(msg: String, sec: Double): Unit = {
  print(msg + ": ")
  println(sec + " seconds")
}

def report(n: Long): Unit = {
  show("Long " + n, timeLong(n))
  if (n <= Int.MaxValue) show("Int  " + n, timeInt(n.toInt))
}

// HUVUDPROGRAM, mätningar:

report(Int.MaxValue)

for (i <- 1 to 10) {
  report(4.26e9.toLong)
}
```

e) Hur mycket snabbare går det att räkna med Int-variabler jämfört med Long-variabler?

## 1.10.2 Frivilliga extrauppgifter

**Uppgift 7.** Lek med färg.

**Uppgift 8.** Ladda ner dessa pdf-kompendier och gör några uppgifter som du tycker verkar intressanta:

a) "Uppdrag med Kojo" som kan laddas ner här:

[fileadmin.cs.lth.se/cs/Personal/Bjorn\\_Regnell/uppdrag.pdf](http://fileadmin.cs.lth.se/cs/Personal/Bjorn_Regnell/uppdrag.pdf)

b) "Programming Fundamentals with Kojo" som kan laddas ner här:

[wiki.kogics.net/kojo-codeactive-books](http://wiki.kogics.net/kojo-codeactive-books)



# Kapitel 2

## Kodstrukturer

- samling: Range
- for-uttryck
- map
- foreach
- flatMap
- algoritm vs implementation
- pseudokod
- algoritm: swap
- algoritm: summering
- algoritm: min/max
- paket
- import
- filstruktur
- jar
- dokumentation
- programlayout
- JDK
- konstanter vs föränderlighet
- objektorientering
- klasser
- objekt
- punktnotation
- referensvariabler
- referenstilldelning
- anropa metoder
- block
- namnsynlighet
- namnöverskuggning
- SimpleWindow

## 2.1 Övning: programs

### Mål

- 

### Förberedelser

- Studera teorin i kapitel 2.
- Bekanta dig med grundläggande terminalkommandon; se appendix A.
- Bekanta dig med den editor du vill använda; se appendix B.

### 2.1.1 Grunduppgifter

**Uppgift 1.** Skapa med hjälp av en editor en fil med namn `hello-script.scala` som innehåller denna enda rad:

```
println("hej skript")
```

Spara filen och kör kommandot `scala hello-script.scala` i terminalen:

```
> scala hello-script.scala
```

- a) Vad händer?
- b) Ändra i filen så att högerparentesen saknas. Spara och kör skriptfilen igen. Vad händer?

**Uppgift 2.** Skapa med hjälp av en editor en fil med namn `hello-app.scala`.

```
> gedit hello-app.scala &
```

Skriv dessa rader i filen:

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

- a) Kompilera med `scalac hello-app.scala` och kör koden med `scala Hello`.

```
> scalac hello-app.scala  
> ls  
> scala Hello
```

Vad heter filerna som kompilatorn skapar?

- b) Vilket alternativ går snabbast att köra igång, ett skript eller en kompilerad applikation? Varför? Vilket alternativ kör snabbast när väl exekveringen är igång?





c) Ändra i din kod så att kompilatorn ger följande felmeddelande:  
Missing closing brace

**Uppgift 3.** Skapa med hjälp av en editor en fil med namn `Hi.java`.

```
> gedit Hi.java &
```

Skriv dessa rader i filen:

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

Kompilera med `javac Hi.java` och kör koden med `java Hi`.

```
> javac Hi.java
> ls
> java Hi
```

- a) Vad heter filen som kompilatorn skapat?
- b) Vad händer om källkodsfilen och klassnamnet inte överensstämmer?

## 2.1.2 Extrauppgifter: öva mer på grunderna

**Uppgift 4.**

## 2.1.3 Fördjupningsuppgifter: avancerad nivå

**Uppgift 5.**



# Kapitel 3

## Funktioner, Objekt

- parameter
- returtyp
- värdeandrop
- namnanrop
- namngivna parametrar
- aktiveringspost
- rekursion
- basfall
- anropsstacken
- objektheapen
- objekt
- modul
- lazy val
- aritmetik
- slumpstal

## 3.1 Övning: functions

### Mål

- 

### Förberedelser

- 

### 3.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 3.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 3.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

## 3.2 Laboration: simplewindow

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 3.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.

### 3.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.



# Kapitel 4

## Datastrukturer

- tupler
- case-klasser
- case-object
- enum i java ???
- Array
- Map
- List
- Vector
- föränderlighet
- iterering
- vektorer i Java vs Scala
- Complex
- Rational
- läsa/skriva textfiler
- Source.fromFile
- java.nio.file

## 4.1 Övning: data

### Mål

- 

### Förberedelser

- 

### 4.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 4.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 4.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.



## 4.2 Laboration: textfiles

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 4.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

### 4.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.



# Kapitel 5

## Vektoralgoritmer

- vektoralgoritmer
- min/max
- strängar
- registrering
- java System.out.println
- Scanner

## 5.1 Övning: vectors

### Mål

- 

### Förberedelser

- 

### 5.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 5.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 5.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

## 5.2 Laboration: cardgame

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 5.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

### 5.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.



# Kapitel 6

## Klasser, Likhet

- klasser
- klassparameter
- primär konstruktor
- alternativa konstruktörer
- referenslikhet
- strukturelikhet
- eq vs ==
- compareTo
- Shape
- Point
- Rectangle

## 6.1 Övning: classes

### Mål

- 

### Förberedelser

- 

### 6.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 6.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 6.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.



## 6.2 Laboration: shapes

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 6.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

### 6.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.



# Kapitel 7

## Arv, Gränssnitt

- klasser
- arv
- polymorfism
- likhet
- equals
- accessregler
- private
- public
- protected
- private[this]
- trait
- inmixning
- Any
- AnyVal
- AnyRef
- Nothing

## 7.1 Övning: traits

### Mål

- 

### Förberedelser

- 

### 7.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 7.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 7.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

## 7.2 Laboration: turtlerace-team

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 7.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

### 7.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.



# Kapitel 8

## Mönster, Undantag

- match
- Option
- null
- try
- catch
- Try
- unapply

## 8.1 Övning: matching

### Mål

- 

### Förberedelser

- 

### 8.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 8.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 8.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.



## 8.2 Laboration: chords - team

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 8.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

### 8.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.



# Kapitel 9

## Matriser

- matriser
- nästlade for-satser
- designexempel: Tre-i-rad
- matriser i Java vs Scala

## 9.1 Övning: matrices

### Mål

- 

### Förberedelser

- 

### 9.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 9.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 9.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

## 9.2 Laboration: maze

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 9.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.

### 9.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.



# Kapitel 10

## Sökning, Sortering

- linjärsökning
- binärsökning
- insättningssortering
- urvalssortering
- sortering till ny vektor
- sortering på plats
- algoritmisk komplexitet

## 10.1 Övning: sorting

### Mål

- 

### Förberedelser

- 

### 10.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 10.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 10.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.



## 10.2 Laboration: surveydata-team

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 10.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

### 10.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.



# Kapitel 11

## Scala och Java

- skillnader mellan Scala och Java
- for-sats i Java
- java for-each i Java
- `ArrayList<Integer>`
- `scala.collection.JavaConversions`
- autoboxing i Java
- primitiva typer i Java
- wrapperklasser i Java

## 11.1 Övning: scalajava

### Mål

- 

### Förberedelser

- 

### 11.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 11.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 11.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

## 11.2 Laboration: scalajava-team

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 11.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.

### 11.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.



# Kapitel 12

## Trådar, Web, Android

- Thread
- Future
- HTML
- Javascript
- css
- Scala.js
- Android

## 12.1 Övning: threads

### Mål

- 

### Förberedelser

- 

### 12.1.1 Grunduppgifter

#### Uppgift 1.

a)

### 12.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

### 12.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.



## 12.2 Laboration: life

### Mål

- Att lära sig.

### Förberedelser

- Att göra.

### 12.2.1 Obligatoriska uppgifter

**Uppgift 1.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.

### 12.2.2 Frivilliga extrauppgifter

**Uppgift 2.** En labbuppgiftsbeskrivning.

- En underuppgift.
- En underuppgift.



# Kapitel 13

## Design

-



# Kapitel 14

## Tentaträning

-



# **Del III**

## **Appendix**





# Appendix A

## Terminalfönster och kommandoskal

### A.1 Vad är ett terminalfönster?

I ett terminalfönster kan man skriva kommandon som kör program och hanterar filer på din dator. När man programmerar använder man ofta terminalkommando för att kompilera och exekvera sina program. Man kan använda terminalkommandon för att navigera och manipulera filerna på datorns disk.

#### Terminal i Linux

#### PowerShell i Microsoft Windows

Microsoft Windows är inte Unix-baserat, men i kommandotolken PowerShell finns alias definierat för en del vanliga unix-kommandon. Du startar Powershell t.ex. genom att trycka på Windows-knappen och skriva powershell.

#### Terminal i Apple OS X

Apple OS X är ett Unix-baserat operativsystem. Många kommandon som fungerar under Linux fungerar också under Apple OS X.

### A.2 Några viktiga terminalkommando

Tipsa om [ss64.com](http://ss64.com)



# **Appendix B**

## **Editera**

**B.1 Vad är en editor?**

**B.2 Välj editor**



# Appendix C

## Kompilera och exekvera

### C.1 Vad är en kompilator?

### C.2 Java JDK

#### C.2.1 Installera Java JDK

### C.3 Scala

#### C.3.1 Installera Scala-kompilatorn

### C.4 Read-Evaluate-Print-Loop (REPL)

För många språk, t.ex. Scala och Python, finns det en interaktiv tolk som gör det möjligt att exekvera enstaka programrader och direkt se effekte. En sådan tolk kallas Read-Evaluate-Print-Loop eftersom den läser en rad i taget och översätter till maskinkod som körs direkt.

#### C.4.1 Scala REPL

##### Kommandon i REPL

`:paste`

Kortkommandon: Ctrl+K etc.



# **Appendix D**

## **Dokumentation**

**D.1 Vad gör ett dokumentationsverktyg?**

**D.2 scaladoc**

**D.3 javadoc**





# Appendix E

## Integrerad utvecklingsmiljö

### E.1 Vad är en IDE?

### E.2 Kojo

#### E.2.1 Installera Kojo

[www.kogics.net/kojo-download](http://www.kogics.net/kojo-download)

#### E.2.2 Använda Kojo

Tabell E.1: Några av sköldpaddans funktioner. Se även [lth.se/programmera](http://lth.se/programmera)

<i>Svenska</i>	<i>Engelska</i>	<i>Vad händer?</i>
fram	forward	Paddan går 25 steg frammåt.
fram(50)	forward(50)	Paddan går 50 steg frammåt.
höger	right	Paddan vrider sig 90 grader åt höger.
upprepa(10){???}	repeat(10){???}	Repetition av ??? 10 gånger.

Koden för den svenska paddans api finns här: [bitbucket.org/lalit\\_pant/kojo/](http://bitbucket.org/lalit_pant/kojo/)

### E.3 Eclipse och ScalaIDE

#### E.3.1 Installera Eclipse och ScalaIDE

#### E.3.2 Använda Eclipse och ScalaIDE



# **Appendix F**

## **Byggverktyg**

**F.1 Vad gör ett byggverktyg?**

**F.2 Byggverktyget sbt**

**F.2.1 Installera sbt**

**F.2.2 Använda sbt**



# **Appendix G**

## **Versionshantering och kodlagring**

### **G.1 Vad är versionshantering?**

### **G.2 Versionshanteringsverktyget git**

#### **G.2.1 Installera git**

#### **G.2.2 Använda git**

### **G.3 Vad är nyttan med en kodlagringsplats?**

### **G.4 Kodlagringsplatsen GitHub**

#### **G.4.1 Installera klienten för GitHub**

#### **G.4.2 Använda GitHub**

### **G.5 Kodlagringsplatsen Atlassian BitBucket**

#### **G.5.1 Installera SourceTree**

#### **G.5.2 Använda SourceTree**



# Appendix H

## Nyckelord

### H.1 Vad är ett nyckelord ord?

Nyckelord är ord som har speciell betydelse och som gör det lättare för kompilatorn att tolka källkoden, eftersom nyckelorden är reserverade för endast ett användningsområde. Man kan till exempel inte använda nyckelordet **def** som namn på en variabel. Nyckelord ges ofta en speciell färg av de kodeditorer som har så kallad syntaxstyrd färgning.

Läs mer här:

[en.wikipedia.org/wiki/Reserved\\_word](https://en.wikipedia.org/wiki/Reserved_word)

#### Nyckelord i Scala

<b>abstract</b>	<b>case</b>	<b>catch</b>	<b>class</b>	<b>def</b>
<b>do</b>	<b>else</b>	<b>extends</b>	<b>false</b>	<b>final</b>
<b>finally</b>	<b>for</b>	<b>forSome</b>	<b>if</b>	<b>implicit</b>
<b>import</b>	<b>lazy</b>	<b>macro</b>	<b>match</b>	<b>new</b>
<b>null</b>	<b>object</b>	<b>override</b>	<b>package</b>	<b>private</b>
<b>protected</b>	<b>return</b>	<b>sealed</b>	<b>super</b>	<b>this</b>
<b>throw</b>	<b>trait</b>	<b>try</b>	<b>true</b>	<b>type</b>
<b>val</b>	<b>var</b>	<b>while</b>	<b>with</b>	<b>yield</b>

#### Nyckelord i Java

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords `const` and `goto` are reserved, even though they are not currently used. `true`, `false`, and `null` might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

<b>abstract</b>	<b>continue</b>	<b>for</b>	<b>new</b>	<b>switch</b>		
<b>assert</b>	<b>***</b>	<b>default</b>	<b>goto</b>	<b>*</b>	<b>package</b>	<b>synchronized</b>
<b>boolean</b>	<b>do</b>	<b>if</b>	<b>private</b>	<b>this</b>		
<b>break</b>	<b>double</b>	<b>implements</b>	<b>protected</b>	<b>throw</b>		

```
byte else import public throws
case enum **** instanceof return transient
catch extends int short try
char final interface static void
class finally long strictfp ** volatile
const * float native super while
*      not used
**      added in 1.2
***      added in 1.4
****      added in 5.0
```



# **Appendix I**

## **Lösningsförslag till övningar**

## I.1 expressions

### I.1.1 Grunduppgifter

#### Uppgift 1.

- a) 42
- b) Lösningstext.

### I.1.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

- a) 42
- b) Lösningstext.

### I.1.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

- a) 42
- b) Lösningstext.

## **I.2** programs

### **I.2.1** Grunduppgifter

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.2.2** Extrauppgifter: öva mer på grunderna

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.2.3** Fördjupningsuppgifter: avancerad nivå

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## I.3 functions

### I.3.1 Grunduppgifter

#### Uppgift 1.

- a) 42
- b) Lösningstext.

### I.3.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

- a) 42
- b) Lösningstext.

### I.3.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

- a) 42
- b) Lösningstext.

## **I.4 data**

### **I.4.1 Grunduppgifter**

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.4.2 Extrauppgifter: öva mer på grunderna**

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.4.3 Fördjupningsuppgifter: avancerad nivå**

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## **I.5** vectors

### **I.5.1** Grunduppgifter

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.5.2** Extrauppgifter: öva mer på grunderna

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.5.3** Fördjupningsuppgifter: avancerad nivå

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## **I.6** classes

### **I.6.1** Grunduppgifter

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.6.2** Extrauppgifter: öva mer på grunderna

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.6.3** Fördjupningsuppgifter: avancerad nivå

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## **I.7** traits

### **I.7.1** Grunduppgifter

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.7.2** Extrauppgifter: öva mer på grunderna

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.7.3** Fördjupningsuppgifter: avancerad nivå

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.



## **I.8** matching

### **I.8.1 Grunduppgifter**

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.8.2 Extrauppgifter: öva mer på grunderna**

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.8.3 Fördjupningsuppgifter: avancerad nivå**

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## I.9 matrices

### I.9.1 Grunduppgifter

#### Uppgift 1.

- a) 42
- b) Lösningstext.

### I.9.2 Extrauppgifter: öva mer på grunderna

#### Uppgift 2.

- a) 42
- b) Lösningstext.

### I.9.3 Fördjupningsuppgifter: avancerad nivå

#### Uppgift 3.

- a) 42
- b) Lösningstext.

## **I.10**    sorting

### **I.10.1**    Grunduppgifter

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.10.2**    Extrauppgifter: öva mer på grunderna

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.10.3**    Fördjupningsuppgifter: avancerad nivå

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## **I.11 scalajava**

### **I.11.1 Grunduppgifter**

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.11.2 Extrauppgifter: öva mer på grunderna**

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.11.3 Fördjupningsuppgifter: avancerad nivå**

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.

## **I.12 threads**

### **I.12.1 Grunduppgifter**

#### **Uppgift 1.**

- a) 42
- b) Lösningstext.

### **I.12.2 Extrauppgifter: öva mer på grunderna**

#### **Uppgift 2.**

- a) 42
- b) Lösningstext.

### **I.12.3 Fördjupningsuppgifter: avancerad nivå**

#### **Uppgift 3.**

- a) 42
- b) Lösningstext.



# **Appendix J**

## **Ordlista**